UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Heber Fernandes Amaral

Estratégia do Aprimoramento adiado para buscas locais: Usando características de ótimos locais como critério na seleção de soluções aprimorantes

Belo Horizonte
2022

Heber Fernandes Amaral

Estratégia do Aprimoramento adiado para buscas locais: Usando características de ótimos locais como critério na seleção de soluções aprimorantes

Versão Final

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Orientador: Sebastián Alberto Urrutia

Belo Horizonte
2022

Amaral, Heber Fernandes Amaral

A485e Estratégia do aprimoramento adiado para buscas locais: [recurso eletrônico] usando características de ótimos locais como critério na seleção de soluções aprimorantes / Heber Fernandes Amaral. – 2022.
1 recurso online (87 f. il, color.) : pdf.

Orientador: Sebastián Alberto Urrutia.

Tese (Doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciências da Computação.
Referências: f. 83-87

1. Computação – Teses. 2. Otimização combinatória – Teses. 3. Programação heurística – Teses. 4. Problema do caixeiro viajante– Teses. I. Urrutia, Sebastián Alberto. II Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Computação. III.Título.

CDU 519.6*61(043)

Ficha catalográfica elaborada pela bibliotecária Irenquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

ESTRATÉGIA DO APRIMORAMENTO ADIADO PARA BUSCAS LOCAIS: USANDO CARACTERÍSTICAS DE ÓTIMOS LOCAIS COMO CRITÉRIO NA SELEÇÃO DE SOLUÇÕES APRIMORANTES

HEBER FERNANDES AMARAL

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. Sebastián Alberto Urrutia - Orientador
Departamento de Ciência da Computação - UFMG

Prof. Thiago Ferreira de Noronha
Departamento de Ciência da Computação - UFMG

Prof. Vinícius Fernandes dos Santos
Departamento de Ciência da Computação - UFMG

Prof. Maurício Cardoso de Souza
Departamento de Engenharia de Produção - UFMG

Prof. Túlio Toffolo
Departamento de Computação - UFOP

Prof. Daniel Aloise
Department of Computer Engineering and Software Engineering - Polytechnique Montréal

Belo Horizonte, 7 de Janeiro de 2022.

À minha esposa e aos meus filhos, por tudo que representam para mim. À Virgem Maria, por sua intercessão. A Deus, por tudo.

Agradecimentos

Registro em primeiro lugar um agradecimento especial a minha amada esposa Carla, pelo imenso apoio e compreensão nesses anos. Por ser o meu norte e meu porto seguro. Em especial nos 3 meses que passei na Noruega.

Ao meu filho, João Lucas, pela compreensão de minhas ausências e pela força e alegria transmitidas nos momentos em que mais precisei.

Ao meu filho/filha ainda em gestação que tanta força já me transmite nestes momentos finais do doutorado.

Ao professor Sebastián, pela orientação extremamente competente, pelo apoio, confiança em mim depositada e por esses anos de aprendizado.

Ao professor Lars M Havttum, pela orientação, apoio e grande aprendizado proporcionado no período em que estive em Molde, Noruega.

Ao Padre Khiem Duc Nguyen e aos amigos Clay e Bright da Igreja Católica de Molde (Noruega) pelo apoio fraternal, espiritual e amizade nos meses em que convivemos.

Aos meus pais por investirem em minha educação de forma insistente e pela formação moral sólida proporcionada.

A minha sogra, Lucia, por todo apoio ao cuidar do meu filho e minha esposa com tanto zelo em minhas ausências.

Aos amigos do LAPO por todo apoio e amizade nesse anos no laboratório. Em especial à Evelyn por toda ajuda neste trabalho.

Aos professores do DCC-UFMG por todo aprendizado proporcionado.

Aos servidores do DCC-UFMG por toda dedicação e zelo que conduzem seus trabalhos.

"*Ninguém é suficientemente perfeito, que não possa aprender com o outro e, ninguém é totalmente destituído de valores que não possa ensinar algo ao seu irmão.*"

(São Francisco de Assis)

Resumo

Busca local é um método heurístico para encontrar boas soluções para problemas combinatórios, baseado em busca em vizinhança. A busca local necessariamente parte de uma solução inicial viável e, através de operações, tenta melhorá-la, gerando novas soluções viáveis, chamadas de soluções vizinhas. Selecionam-se dentre as soluções vizinhas, soluções melhores do que a solução corrente por algum critério preestabelecido. Esse processo é repetido até que não seja possível obter nenhuma solução aprimorante. A solução final de uma busca local é chamada de ótimo local, a melhor solução possível para um problema é chamada de ótimo global.

Existem várias estratégias para implementar buscas locais, sendo as mais conhecidas as abordagens Melhor Aprimorante (*Best Improvement*) e Primeiro Aprimorante (*First Improvement*). A abordagem Melhor Aprimorante seleciona a melhor solução gerada pela função de vizinhança a cada iteração, desde que seja melhor que a solução corrente. A abordagem Primeiro Aprimorante seleciona a primeira solução melhor do que a solução corrente (aprimorante) gerada pela função de vizinhança.

A tese propõe uma nova abordagem de Busca Local denominada Melhoria Adiada (*Delayed improvement*). Tal abordagem tenta evitar ótimos locais de baixa qualidade. Para isso, seleciona a cada iteração um vizinho aprimorante com poucos atributos de um ótimo local usando desigualdades baseadas em cortes, que normalmente são usadas em modelos de Programação Linear Inteira. Este método foi implementado e testado usando o problema do Caixeiro Viajante e o problema do Corte Máximo como casos de uso. Para o problema do Caixeiro Viajante foi usada a busca local *2-OPT* como referência e para o problema do Corte Máximo foi usada a busca local 1-Flip como referência. Os resultados computacionais, embora mais lentos, obtiveram ótimos locais melhores quando comparados com as estratégias convencionais. A nova estratégia obteve melhores resultados comparativamente em experimentos com tempo de computação fixo ou com valor alvo fixo da função objetivo.

Palavras-chave: otimização combinatória; busca local; heurísticas; PCV; *2-OPT*.

Abstract

Local search is a neighbourhood based heuristic method, that aims to find good solutions for combinatorial problems. A Local search starts with a viable solution and through operations attempts to improve the initial solution, generating new viable solutions. The new solutions are called the neighbourhood of the current solution. Among neighbouring solutions is selected a better solution than the current solution by some pre-established criteria. Then, the selected solution becomes the new current solution creating a new neighbourhood. This process repeats until no better solution can be obtained among neighbouring solutions. The final solution of a local search is a local optimum. The best possible solution of a problem is a global optimum.

There are several strategies for implementing local searches; the best known are Best Improvement and First Improvement. The Best Improvement approach selects, at each iteration, the best solution generated by the neighbourhood function; if it is better than the current solution. The First Improvement approach selects the first genereted solution, better than the current solution.

This thesis presents a new approach to Local Search named Dalayed Improvement. Such approach attempts to avoid low quality local optimum. It tries to select in each iteration a neighbour soluttion that, been better than the currente one, it has less attributes of a local optimum. This approach is based on cuts (inequalities), commonly used in Integer Programming models. This method was implemented and tested using the Travelling Salesman Problem (TSP) and Max-Cut Problem as case studies. For TSP we used 2-OPT local search as reference and for Max-Cut Problem we used 1-Flip local search as reference. The computational results, although slower, had better local optimum when compared to conventional strategies. The new strategy obtained better results compared to experiments with fixed computation time or with a fixed target value for the objective function.

Keywords: combinatorial optimization; local search; heuristics; TSP; 2-OPT.

Lista de Figuras

*

Lista de Tabelas

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

Problemas de otimização combinatória surgem em diferentes áreas da ciência da computação e em áreas que utilizam métodos computacionais. Esses problemas geralmente precisam encontrar agrupamentos, ordenações ou arranjos de um conjunto finito e discreto de objetos que satisfaçam certas condições ou restrições.

Uma instância de um problema de otimização combinatória possui um conjunto de soluções viáveis e uma função de custo sobre as soluções. O problema consiste em encontrar a melhor solução, chamada de *solução ótima*, entre todas as soluções possíveis [6]. Encontrar essa solução pode ser uma tarefa extremamente difícil do ponto de vista computacional.

Os métodos usados para resolver problemas de otimização combinatória dividem-se em dois grandes grupos: métodos exatos, cujo objetivo é encontrar a solução ótima de um problema; e os métodos heurísticos, que buscam encontrar a melhor solução possível dentro de um determinado tempo considerado viável. Um método heurístico bastante utilizado é chamado de *busca local*, que é o foco deste trabalho.

A busca local é fundamentada no conceito de vizinhança. O processo de busca local considera uma solução inicial viável para o problema e evolui através de operações sobre essa solução com o objetivo de obter melhores soluções viáveis. As soluções geradas a partir de operações sobre uma solução qualquer s são chamadas de soluções vizinhas de s. Selecionam-se, dentre as soluções vizinhas, soluções melhores do que a solução corrente por algum critério preestabelecido. Esse processo é repetido até que a iteração não gere nenhuma solução aprimorante, ou seja, até que a busca convirja. A solução final de uma busca local é chamada de ótimo local.

A presente tese propõe uma nova abordagem de busca local, que tenta evitar ótimos locais de baixa qualidade, selecionando a cada iteração um vizinho aprimorante com poucos atributos de um ótimo local. A abordagem utiliza como parâmetro desigualdades baseadas em cortes, que normalmente são usados em modelos de programação linear inteira. O resultados desse trabalho foram publicados em junho de 2021 no artigo *Delayed improvement local search* na revista *Journal of Heuristics* [5].

1.1 Definições

Nessa seção, são apresentados conceitos básicos importantes para a compreensão do trabalho. A notação adotada segue o padrão apresentado em [10].

Um *grafo* $G = (V, E)$ é representado por um conjunto não vazio de *vértices* V e um conjunto de *arestas* E definidas por um par não ordenado de vértices $\{\{u, v\} \in E | u, v \in V\}$. Os vértices do par formam uma aresta e são definidos como suas *extremidades*, dizemos que uma aresta é *incidente* a suas *extremidades*. Tem-se um *laço* quando uma aresta conecta um vértice a ele mesmo. Um vértice u é dito *adjacente* a outro vértice v se o grafo G contém uma aresta (u, v).

Definição 1 (**Grafo simples**[1]). *Grafo que não possui laços e há no máximo uma única aresta entre um mesmo par de vértices.*

Definição 2 (**Grafo completo**). *Grafo no qual existe uma aresta entre quaisquer dois vértices distintos. Um grafo completo com n vértices é denotado por* K_n.

Definição 3 (**Grafo orientado**). *Um grafo orientado* $G = (V, A)$ *é definido por um conjunto finito* V *de vértices e um conjunto de pares ordenados* A *que representam arestas orientadas, também chamadas de arcos,* $\{(u, v) \in A | u, v \in V\}$ *que conectam certos pares de vértices.*

Definição 4 (**Grafo orientado ponderado**). *Um grafo orientado ponderado* $G = (V, A, w)$ *é definido por um conjunto finito* V *de vértices, um conjunto de pares ordenados* A *que representam arestas orientadas (arcos)* $\{(u, v) \in A | u, v \in V\}$ *e* $w : A \mapsto \mathbb{R}^+$ *uma função que atribui a cada arco* $a \in A$ *um peso* $w(a)$.

Definição 5 (**Caminho**). *Seja* $G = (V, E, w)$ *um grafo orientado ponderado. Um caminho em* G *é uma lista* $(v_1, v_2, ..., v_k)$ *de vértices* $v_i \in V (i = 1, ..., k)$ *tal que cada par* (v_i, v_{i+1}) *é uma aresta em* G.

Definição 6 (**Ciclo**). *Um ciclo em um grafo* G *é um caminho em* G *cujo o primeiro vértice é igual ao ultimo vértice.*

Definição 7 (**Ciclo Hamiltoniano**). *Um ciclo Hamiltoniano em um grafo* G *é um ciclo em* G *que visita todos os vértice exatamente uma vez (exceto o vértice de partida), i.e.,* $P = (v_1, v_2, ..., v_n, v_1)$ *é um Ciclo Hamiltoniano em* G *se, somente se,* $n = |V|$, *e* $\{v_1, v_2, ..., v_n\} = V$.

[1] No decorrer do texto, quando se tratar deste tipo de grafo, o adjetivo simples será omitido.

Definição 8 (Custo de um caminho ou ciclo). *Dado um grafo orientado e ponderado G e um caminho $p = (v_1, v_2, ..., v_k)$ em G, o custo do caminho $w(p)$ e definido como $w(p) = \sum_{i=1}^{k-1} w((v_i, v_{i+1}))$.*

1.2 Problema do Caixeiro Viajante

Para avaliar a qualidade da abordagem desenvolvida nesse trabalho, foram realizados experimentos utilizando instâncias de um problema clássico da computação chamado de Problema do Caixeiro Viajante (PCV). O PCV é motivado pelo problema do mundo real descrito a seguir: dado um conjunto de cidades, um caixeiro deseja visitar todas as cidades uma única vez com retorno a sua cidade de origem. O problema do Caixeiro Viajante é um dos mais estudados em análise combinatória, foi formulado pela primeira vez em 1954 por Dantzig, Fulkerson e Jonson em [16]. Em 1972, Karp provou que o PCV é um problema NP-difícil [36].

Entre as aplicações práticas do problema do Caixeiro Viajante estão: perfuração de placas de circuitos impressos, cristalografia de raios X, revisão de motores de turbinas a gás, problema de coleta de pedidos em armazéns e roteirização de veículos [35, 48, 41].

É possível definir o PCV utilizando conceitos de teoria dos grafos da seguinte forma: dado um grafo direcionado e ponderado, deseja-se encontrar um ciclo hamiltoniano de menor custo. E é possível construir qualquer instância de um PCV associada a um grafo G não completo sobre um grafo G' completo, tal que a instância do PCV para G' tenha exatamente o mesmo resultado que para G. Isso é feito adicionando em G as arestas faltantes com peso alto o suficiente para que nunca apareçam em uma solução ótima. No restante desse trabalho, assumiremos que as instâncias para o PCV são sempre associadas a um grafo completo. Diante disso, os ciclos Hamiltonianos desses grafos são exatamente as permutações do conjunto de vértices do grafo associado. Chamaremos de rota o ciclo hamiltoniano que é uma solução para o problema do Caixeiro Viajante.

Definição 9 (Problema do Caixeiro Viajante (PCV)). *Dado um grafo orientado e ponderado G, o PCV tem como objetivo encontrar o ciclo Hamiltoniano de custo mínimo em G.*

1.3 Problema do Corte Máximo

Outro problema utilizado para avaliar a qualidade da abordagem desenvolvida nesse trabalho foi o problema do corte arestas máximo. O problema da determinação de um corte de arestas máximo pode ser definido da seguinte forma: dado um grafo $G = (V, E)$ e $w : E \mapsto \mathbb{R}^+$ uma função que atribui a cada aresta $e \in E$ um peso $w(e)$, determinar um corte $(S, V \backslash S)$ no conjunto de vértices V, de forma que a soma dos pesos das arestas que ligam S a $V \backslash S$ seja máximo. Quando se atribui um peso unitário a todas as arestas do grafo, o problema se reduz a determinar o corte de aresta de cardinalidade máxima. Também chamado de Problema do Corte Máximo Simples, é conhecido como sendo um problema NP-difícil [21], versão utilizada neste trabalho, que será denotado como Problema do Corte Máximo (MAX-CUT).

1.4 Principais Contribuições deste Trabalho

As principais contribuições desse trabalho são:

- Desenvolvimento de uma estratégia de busca local que evita convergências prematuras.
- Aplicação da nova estratégia de busca local no Problema do Caixeiro Viajante.
- Aplicação da nova estratégia de busca local no Problema do Corte Máximo.
- Estudo do impacto da estratégia de busca local proposta tanto no tempo de busca quanto na qualidade do resultado.

1.5 Organização do Texto

O restante do texto encontra-se organizado da seguinte forma: No Capítulo 2, apresenta-se uma revisão de literatura para melhor compreensão das questões abordadas no presente trabalho. No Capítulo 3, os fundamentos usados para criação da abordagem proposta. No Capítulo 4, a primeira versão da abordagem proposta aplicada ao problema

do Caixeiro Viajante e os resultados obtidos. No Capítulo 5, a versão aprimorada da aplicação da abordagem ao problema do Caixeiro Viajante e os resultados obtidos. No Capítulo 6, a aplicação da abordagem ao problema do Corte Máximo e os resultados obtidos. E no Capítulo 7, sumarizamos o que foi apresentado nos capítulos anteriores e as conclusões são apresentadas.

Capítulo 2

Revisão Bibliográfica

A partir do fim dos anos 50, métodos de busca começaram a ser utilizados em problemas de otimização combinatória. Foi nesse período que o primeiro algoritmo baseado em troca de arestas foi desenvolvido para o PCV. Em seguida, o conceito básico de algoritmo baseado em troca foi aplicado a uma variedade de problemas. Nicholson [1971] ampliou o conceito de estratégias de troca para uma classe mais geral de problemas de permutação, roteamento e estoque. Apesar dos sucessos iniciais, a busca local não foi considerada uma técnica madura por muito tempo, até que, nos anos 90, o interesse pelo assunto aumentou, devido aos seguintes aspectos elencados por [1]:

- Muitas variações de algoritmos de busca local foram propostas baseadas em analogias de processos na natureza e outras áreas do conhecimento. Como, física, estatística, evolução biológica e neurofisiologia. São dessa época: *simulated annealing*, algoritmos genéticos e algumas variantes de redes neurais.

- Alguns dos algoritmos de busca local foram matematicamente modelados, produzindo resultados teóricos sobre seu desempenho. Além disso, [34] introduziram uma teoria da complexidade da busca local que forneceu uma visão mais teórica, não apenas na complexidade da busca local, mas também na estrutura combinatória de problemas de otimização discreta.

- O grande aumento nos recursos computacionais, somado ao uso de estruturas de dados sofisticadas, tornaram os algoritmos de busca local fortes concorrentes dentro da classe de algoritmos projetados para lidar com grandes instâncias de problemas. Além disso, a flexibilidade e a facilidade de implementação de algoritmos de busca local levaram ao tratamento bem-sucedido de muitos problemas complexos do mundo real.

Com a criação de bibliotecas de instâncias padronizadas, houveram avanços também no campo experimental. Isso facilitou a comparação de resultados entre diferentes abordagens. Um exemplo foi a criação da TSPLIB proposta por [50], que contém mais de 110 instâncias do PCV, algumas com até 85.900 cidades, incluindo instâncias originarias de circuitos impressos e aplicações *VLSI (Very Large Scale Integration)*.

De 1973 até o fim da década de 80 o título de melhor heurística para resolução do PCV era atribuído ao algoritmo [43]. O algoritmo Lin-Kernighan é uma generalização do 3-OPT e outras ideias anteriores dos mesmos autores, que se assemelham com a busca tabu. Em 2000 essa abordagem ganhou novo fôlego com a publicação de [30] que descreveu uma versão modificada do algoritmo Lin-Kernighan. Os resultados mostraram que a versão apresentada por [30] encontrava os ótimos globais de todas as instâncias do TSPLIB em que o ótimo global era conhecido na época de sua publicação. Em 2003 [7] fizeram modificações no algoritmo Lin-Kernighan que permitiram a resolução de instâncias muito grandes. Os resultados mostrados por [7] resultaram em no máximo 1% de erro em relação ao ótimo global estimado para instâncias com até 25.000.000 de cidades.

A Busca Local pode encontrar soluções satisfatórias muito rapidamente. No entanto, tende a ficar presa em ótimos locais de baixa qualidade. Para melhorar a eficácia da Busca Local e evitar tal ocorrência, várias técnicas foram introduzidas ao longo dos anos. Nesse sentido, também foram publicados diversos trabalhos utilizando e descrevendo metaheurísticas baseadas em busca local, destacando-se: *Variable Neighborhood Search (VNS)*, *Variable Neighborhood Descent(VND)*, *Iterated Local Search (ILS)*, *Simulated Annealing*, *Tabu Seach* e *Greed Randomized Adaptive Search Procedures (GRASP)*, sobre os quais o interesse cresceu rapidamente. O que é evidenciado pelo número crescente de artigos publicados a cada ano sobre este tópico.

Descreveremos a seguir as abordagens baseadas em Busca Local mais relevantes para o presente trabalho.

2.1 *Guided Local Search (GLS)*

Guided Local Search (GLS) ou Busca Local Guiada é uma variação da Busca Local clássica proposta por [55]. Os princípios do *GLS* podem ser resumidos da seguinte forma: Para aplicar o *GLS*, define-se um conjunto de características para as soluções candidatas; Quando a busca encontra um ótimo local, certas características são selecionadas e penalizadas. Então, a busca local continua, desse ponto em diante, usando a função objetivo alterada pelas penalidades acumuladas. O *GLS*, na prática, distribui o esforço de busca favorecendo áreas promissoras. A ideia por trás dos métodos de *GLS* é o uso de informações prévias e adquiridas durante a busca para sair de ótimos locais. Dada uma função objetivo g o *GLS* define uma nova função h, como mostrado abaixo: [4]

$$h(s) = g(s) + \lambda \times \sum_{i \in Caracteristicas} (p_i \times I_i(s))$$

A função h será usada pela busca local substituindo a função objetivo original g, sendo: s a solução candidata, λ um parâmetro do método, p_i a penalidade para característica i e I_i uma função que retorna o valor 1 se s possuir a característica i e 0 caso contrário.

A intenção do *GLS* é penalizar características desfavoráveis ou características que conduzem a um ótimo local. A utilidade de uma característica i, $util_i$, para um ótimo local s, é definida da seguinte forma:

$$util_i = I_i(s) \times \frac{c_i}{1 + p_i}$$

Onde c_i é o custo da característica i e p_i é a penalidade para a característica i.

O algoritmo 2.1 descreve a implementação mais comum do *GLS*. Como pode ser visto, o *GLS* começa com uma solução inicial s e a cada iteração aplica-se uma busca local na solução corrente gerando um novo ótimo local s'. Então, verifica-se a utilidade de cada característica em s' e penalizam-se as características que possuem maior utilidade. Ao final retorna-se o melhor ótimo local encontrado.

Algoritmo 2.1: Busca Local Guiada (*Guided Local Search, GLS)*

```
1  GuideLocalSearch(s,g,λ,I[],c[],Características){
2  k = 1;
3  h = g + λ * ∑(p_i * I_i);
4  s* = s;
5  enquanto Critério de parada faça
6  |  s' = BuscaLocal(s,h);    // Usa a função h na avaliação da vizinhança
7  |  para cada i ∈ Características faça
8  |  |  util_i = I_i(s') * c_i/(1 + p_i);
9  |  fim
10 |  para cada i ∈ Características tal que util_i é máximo faça
11 |  |  p_i = p_i + 1;
12 |  fim
   |  // Em caso de minimização
13 |  se g(s') ¡ g(s*) então
14 |  |  s* = s';
15 |  fim
16 |  s = s';
17 fim
18 retorna s*;
19 }
```

Com relação a *GLS*, podemos citar os trabalhos [3, 14, 53, 12, 49], a se destacar [8] que implementaram três operadores de busca locais para resolução do problema de roteamento de veículos (*vehicle routing problem (VRP)*): Um operador baseado no algoritmo Lin-Kernighan para otimização da rota de cada veículo, um operador baseado no algoritmo *CROSS-exchange* para otimização da divisão das rotas e um terceiro

operador chamado pelo autor de *Relocation Chain*. A solução foi feita em uma estrutura de busca local guiada (GLS). Para detectar soluções de baixa qualidade com mais precisão, usou *insights* de um estudo exploratório sobre as propriedades das soluções de VRP. A heurística resultante calcula soluções de alta qualidade para uma ampla gama de instâncias de *benchmark*. Demonstram também que a heurística pode ser usada para resolver outras variantes do VRP, como o VRP com múltiplos depósitos e o VRP com múltiplas viagens.

2.2 Busca em Vizinhança Variável

A Busca em Vizinhança Variável (*Variable Neighborhood Search, VNS*), proposta por [47], é um método de busca local que consiste em explorar o espaço de soluções por meio de trocas sistemáticas de estruturas de vizinhança. Diferentemente de outras estratégias de busca local, o método VNS não segue uma trajetória, mas explora diferentes vizinhanças gradativamente. O método inclui, também, um procedimento de busca local a ser aplicado sobre a solução corrente. Esta rotina de busca local também pode usar diferentes estruturas de vizinhança [26, 27].

Algoritmo 2.2: Busca em Vizinhança Variável (*Variable Neighborhood Search, VNS*)

```
1  VNS(s,K_max){
2  enquanto Critério de parada faça
3  |  k = 1;
4  |  enquanto k < K_max faça
5  |  |  s' = N^(k)(s);
6  |  |  s" = BuscaLocal(s');
7  |  |  se g(s") ¡ g(s') então
8  |  |  |  s = s";
9  |  |  |  k = 1;
10 |  |  senão
11 |  |  |  k = k+1;
12 |  |  fim
13 |  fim
14 fim
15 retorna s;
16 }
```

O algoritmo 2.2 descreve o caso clássico do VNS. Nesse algoritmo o VNS começa com uma solução inicial s e a cada iteração seleciona-se um vizinho s' qualquer dentro da

vizinhança $N^{(k)}(s)$ da solução s corrente. Então s' é submetido a um procedimento de busca local retornado s". Se s" for melhor que a solução s corrente, a busca continua de s" recomeçando da primeira vizinhança $N^{(1)}(s)$. Caso contrário, continua-se a busca a partir da próxima vizinhança $N^{(k+1)}(s)$. Esse procedimento é encerrado quando uma condição de parada for atingida, podendo ser: o tempo máximo permitido de CPU, o número máximo de iterações ou número máximo de iterações consecutivas sem melhoramentos. [26, 27]

O VNS vem ganhando destaque no meio acadêmico desde sua criação. Segundo [29] foram mais 250 publicações em periódicos sobre o VNS só em 2016.

2.3 *Itereted Local Seach (ILS)*

O *Iterated Local Search (ILS)* pretende melhorar o processo de busca local gerando novas soluções de partida que são obtidas por meio de perturbações na solução ótima local. A perturbação precisa mudar a solução atual de forma a permitir que a busca local explore diferentes soluções e, ao mesmo tempo, evite um reinício completo da busca. O método ILS, portanto, foca a busca em um pequeno subespaço definido por soluções que são ótimas locais. [44]

Algoritmo 2.3: Busca Local Iterada (*Iterated Local Search (ILS)*)

```
1 ILS(s_0){
2 s* = LocalSearch(s_0);
3 do
4 |   s' = Pertubacao(s*);
5 |   s*' = LocalSearch(s');
6 |   s* = CriterioDeAceite(s*,s*');
7 until critério de parada;
8 retorna s*;
9 }
```

O algoritmo 2.3 demostra, em linhas gerais, como funciona o ILS. O ILS começa com uma solução inicial s_0. Aplica-se uma busca local em s_0 até parar em um ótimo local $s*$. Em cada iteração $s*$ é perturbado gerando s'. Aplica-se uma busca local em s' gerando $s*$'. Então, verifica-se se $s*'$ pode ser tornar a nova solução corrente $s*$ por uma função de avaliação. O procedimento se repete até que a condição de parada seja atingida, podendo ser: o tempo máximo de CPU, o número máximo de iterações, número máximo de iterações consecutivas sem melhoramentos, entre outros.

2.4 *Simulated Annealing*

Proposta originalmente por [37] e [13], *Simulated Annealing* ou Recozimento Simulado é uma busca local probabilística que foi inspirada no recozimento da metalurgia, uma técnica que envolve aquecimento e resfriamento controlado de um material para aumentar o tamanho de seus cristais e reduzir seus defeitos.

O *Simulated Annealing (SA)* é uma das metaheurísticas mais simples e mais conhecidas para tratar de problemas de otimização de caixa preta, cuja função objetivo não é dada explicitamente. A principal vantagem do SA é a sua simplicidade. Quando a avaliação da função-objetivo resulta de processos de simulação complexos envolvendo muita memória, os algoritmos de base populacional não são aplicáveis e o SA é um boa opção para tratar de tais problemas [17]. Nesse sentido, podemos citar os trabalhos [46, 32, 25, 9].

O *Simulated Annealing* começa com uma solução inicial viável s. O procedimento principal consiste em cada iteração gerar um único vizinho s' da solução corrente s. A solução vizinha s' passará a ser a nova solução corrente se a solução vizinha for aprimorante. Caso não seja aprimorante, a solução vizinha s' também poderá ser aceita, mas neste caso, com uma probabilidade $e^{-\Delta/T}$, onde $\Delta = f(s') - f(s)$, sendo f a função objetivo do problema e T é um parâmetro do método. T é chamado de temperatura e regula a probabilidade de se aceitar soluções não aprimorantes. A temperatura T assume, inicialmente, um valor elevado T_0. Após um número fixo de iterações, a temperatura é gradativamente diminuída por uma razão de resfriamento α, tal que $T_k \leftarrow \alpha T_{k-1}$, sendo 0 ¡ α ¡ 1. Com esse procedimento dá-se uma chance maior para escapar de ótimos locais no inicio. À medida que T aproxima-se de zero, o algoritmo comporta-se como o método de descida, uma vez que diminui a probabilidade de se aceitar movimentos não aprimorantes. A formulação mostrada aqui é a do resfriamento geométrico, mas, há outras formulações como linear ou logarítmico [33].

Segundo [33], experimentalmente observa-se que o *Simulated Annealing* precisa de um tempo longo para obter resultados de boa qualidade. O tempo gasto no início e no final do processo é pouco efetivo. Uma execução mais longa da *Simulated Annealing* tende a produzir melhores resultados que diversas repetições mais curtas.

2.5 Busca Tabu

A Busca Tabu tem origem no trabalho de [23] que introduziu os conceitos-chave sem usar o termo Tabu. A heurística da mochila apresentada em [23] ilustra o uso da memória de curto prazo, mas o nome da Busca Tabu foi cunhado em [24]. Foi também a primeira vez que a metaheurística foi usada na literatura [39]. Dentre as publicações mais recentes utilizando *Tabu Search* podemos citar os trabalhos [58, 56, 54, 52, 51, 42].

A Busca Tabu consiste em explorar o espaço de soluções movendo-se de uma solução corrente para a melhor solução vizinha mesmo que não seja aprimorante. Mais especificamente, começando com uma solução inicial s, a cada iteração, seleciona-se o vizinho s' pertencente a $N(s)$ com melhor valor segundo a função de avaliação, mesmo que tal solução seja pior que s. Esse critério de escolha é utilizado para escapar de ótimos locais. Essa estratégia, entretanto, pode fazer com que o algoritmo retorne a uma solução já gerada anteriormente. Para evitar que isto ocorra, existe uma lista de movimentos proibidos chamada de lista Tabu. A lista tabu clássica contém os movimentos reversos aos últimos N movimentos realizados (onde N é um parâmetro do método). A lista Tabu funciona como uma fila de tamanho fixo, isto é, quando um novo movimento é adicionado à lista, o mais antigo sai. Como pode-se observar, dessa forma, a lista tabu pode proibir movimentos para soluções que ainda não foram visitadas. Por isso, existe também uma função de aspiração, que é um mecanismo que retira, sob certas circunstâncias, o status tabu de um movimento. A Busca Tabu, termina quando é atingido um certo número máximo de iterações sem melhora no valor da melhor solução. Ou quando o valor da melhor solução alcança um limite inferior conhecido. Esse segundo critério evita a execução desnecessária do algoritmo quando uma solução ótima é encontrada ou quando uma solução é julgada suficientemente boa [39].

2.6 *Local Branching*

Essa estratégia é uma metaheurística híbrida que combina técnicas de solução de programação inteira mista (MIP) com conceitos metaheurísticos típicos, como buscas locais, definições de vizinhança e mecanismos de intensificação e diversificação. A tecnica Local Branching usa um solucionador MIP genérico para explorar subespaços (vizinhanças) viáveis que são construídos através da introdução de desigualdades lineares conhecidas como restrições de ramificação local. A técnica começa considerando a

formulação geral de um problema de programação linear binário.

$$\begin{cases} GP = max\ C^t x \\ s.t.\ Ax = b \\ x \in \{0,1\}^n \end{cases}$$

Onde $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times m}$, e $b \in \mathbb{R}^m$. Seja x^0 uma solução viável de GP, $N = \{1, ..., n\}$, $S_0 = \{j \in N | x_j^0 = 0\}$ e k um inteiro positivo. Seja $\tilde{x} = x^0$ uma solução inicial e $bestLB = c^T \tilde{x}$ o *lower bound* inicial. Tendo a desigualdade $\Delta(x, x^0) = \sum_{j \in S_0}(1 - x_j) + \sum_{j \in N \setminus S_0} x_j \leq k$ a região viável do problema GP pode ser dividida pegando, de um lado, os valores de x que satisfazem a inequação $\Delta(x, x^0) \leq k$ e, por outro lado, os valores de x que satisfazem $\Delta(x, x^0) \geq k + 1$.

2.7 Considerações Gerais

Como pode ser visto, com exceção de *Local Branching*, todas as estratégias de busca local detalhadas aqui aceitam soluções vizinhas não aprimorantes. Tal característica tem como objetivo sair de ótimos locais ou evitá-los. A abordagem proposta pelo presente trabalho difere nesse ponto de todas as abordagens apresentadas, pois visa evitar ótimos locais de baixa qualidade garantido um gradiente de melhoria da solução corrente a cada iteração. Não propomos uma nova metaheurística, mas sim, uma nova implementação de busca local.

No Capítulo 3 são detalhadas as características de uma busca local. Serão apresentados também os fundamentos teóricos da abordagem proposta neste trabalho.

2.8 *Time-to-target (TTT plot)*

Os gráficos *Time-to-target* (TTT) [2] exibem a probabilidade de um algoritmo encontrar uma solução pelo menos tão boa quanto um determinado valor alvo em um determinado tempo. Para criar esses gráficos, é definido um custo alvo para a solução de uma instância do problema e é medido o tempo necessário (até um limite de tempo alto) para cada método para chegar a uma solução com um custo tão bom quanto o alvo. O processo é repetido um grande número de vezes e os tempos de execução são classificados

em ordem crescente. Associa-se ao i-ésimo tempo de execução t_i uma probabilidade $p_i = (i - 1/2)/n$, e traçamos os pontos $z_i = [t_i, p_i]$, para $i = 1, ..., n$. Como exemplo mostramos a Figura 2.1, que ilustra um gráfico de distribuição de probabilidade cumulativa para um par instância/alvo obtido aplicando repetidamente uma heurística GRASP para encontrar uma solução com valor de função objetivo pelo menos tão bom quanto um determinado valor alvo. Cada ponto plotado mostra a probabilidade (eixo das ordenadas) da estratégia de alcançar a solução alvo no tempo indicado (eixo das abcissas). Portanto os gráficos *Time-to-target* se mostram boa ferramenta para comparação de abordagens e será usado na análise dos resultados computacionais a serem apresentados nos próximos capítulos.

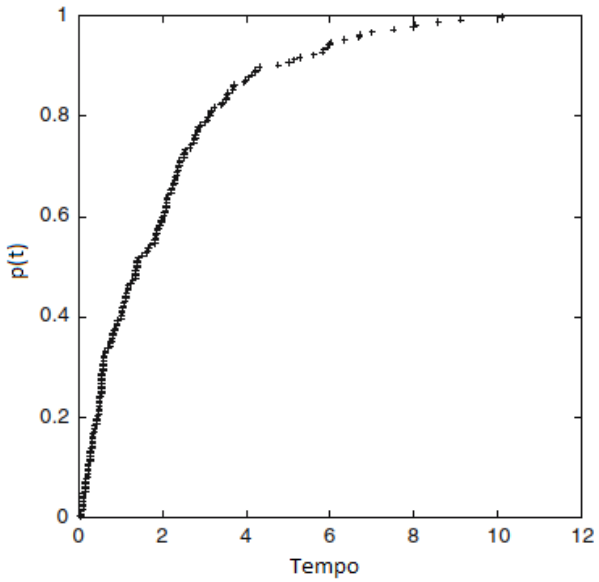

Figura 2.1: Gráfico de distribuição de probabilidade cumulativa, Gráfico *Time-to-target* [2]

Capítulo 3

Busca Local

Neste capítulo serão apresentados conceitos importantes sobre a busca local e os fundamentos teóricos que nos guiaram no desenvolvimento da nova abordagem.

Definição 10 (**Instância de um Problema de Otimização Combinatória**). *Uma instância de um problema de otimização combinatória é um par (ℓ,f) onde ℓ é o conjunto de soluções viáveis e* $f : \ell \mapsto \mathbb{R}$ *uma função de custo. O problema consiste em encontrar o ótimo global, i.e., no caso de problema de minimização, encontrar um* $i^* \in \ell$ *tal que* $f(i^*) \leq f(i)$ *para todo* $i \in \ell$*. Assim,* $f^* = f(i^*)$ *denota o custo ótimo e* $\ell^* = \{i \in \ell | f(i) = f^*\}$ *denota o conjunto de soluções ótimas [1].*

Definição 11 (**Vizinhança**). *Seja (ℓ,f) uma instância de um problema combinatório. Uma função de vizinhança é um mapeamento* $N : \ell \mapsto 2^{\ell}$*, que define para cada solução* $i \in \ell$ *um conjunto* $N(i) \subseteq \ell$*. O conjunto* $N(i)$ *é a vizinhança de* i *e cada* $j \in N(i)$ *é um vizinho de* i*. Assumimos que* $i \in N(i)$*, para todo* $i \in \ell$*. [1]*

Na Definição 10, é descrita uma instância de um problema de minimização, que é o foco deste trabalho, todavia existem também os problemas de maximização, cujo o objetivo é encontrar um $i^* \in \ell$ tal que $f(i^*) \geq f(i)$ para todo $i \in \ell$. Entretanto, nesse trabalho usaremos sempre problemas de minimização, omitindo tal especificação.

Uma busca local tem início com uma solução viável do problema. A função de vizinhança $N(s)$ de uma busca local retorna um conjunto de soluções viáveis "próximas" de s. Ao selecionar uma nova solução s' em $N(s)$ melhor do que s, define-se uma nova vizinhança $N(s')$. Esse processo se repete sucessivamente até que não seja possível encontrar uma solução melhor que a corrente dentro da vizinhança. A solução corrente da última iteração é um ótimo local para aquela vizinhança [40].

Em uma vizinhança $N(s)$, as soluções melhores do que s, são chamadas de *soluções aprimorantes*. Em cada iteração da busca local, podem existir várias soluções aprimorantes possíveis de serem selecionadas como solução corrente para a próxima iteração. A maneira como essa solução será escolhida influencia na qualidade da solução final e no número total de iterações do processo. Existem diversas estratégias de seleção de soluções aprimorantes, por exemplo, escolha aleatória entre os aprimorantes, pior aprimorante, melhor aprimorante (*best improvement*) e primeiro aprimorante (*first improvement*). As

duas últimas são as mais conhecidas e utilizadas. A seguir detalharemos algumas dessa estratégias baseado nas definições de [31].

Na técnica melhor aprimorante, seleciona-se um dos vizinhos que alcançam a maior melhoria na função de avaliação. Formalmente, a função de seleção do melhor aprimorante pode ser definida como: dada uma solução s, seja a função de avaliação das soluções definida como $g(s)$ e $g^\star = \{\min g(s')|s' \in N(s)\}$, assim, $g^\star$ é o melhor valor encontrado para a função de avaliação na vizinhança de s. Seja $I^\star(s) = \{s' \in N(s)|g(s') = g^\star\}$ o conjunto dos melhores vizinhos de s na vizinhança. Pelo método do melhor aprimorante, devemos selecionar para o próximo passo soluções pertencentes a $I^\star$. Percebe-se, dessa forma, que a estratégia melhor aprimorante requer uma completa avaliação de toda vizinhança em cada passo.

A técnica pior aprimorante busca selecionar a cada iteração o vizinho com a pior melhoria na função de avaliação. Formalmente, a abordagem pior aprimorante pode ser definida como: dado uma solução s, seja a função de avaliação das soluções definida como $g(s)$ e $g^\star = \{\max g(s')|s' \in N(s)\ e\ g(s) > g(s')\}$, assim, $g^\star$ é o pior valor aprimorante (cujo $g(s) > g(s')$) encontrado para a função de avaliação na vizinhança de s. Seja $I^\star(s) = \{s' \in N(s)|g(s') = g^\star\}$ o conjunto dos piores vizinhos aprimorantes de s na vizinhança. Pelo método do pior aprimorante, devemos selecionar para o próximo passo soluções pertencentes a $I^\star$. Percebe-se, dessa forma, que a estratégia pior aprimorante, assim como a melhor aprimorante, requer uma completa avaliação de toda vizinhança em cada passo. O objetivo dessa estratégia é garantir o gradiente de melhoria entre as soluções enquanto se faz uma busca mais longa (com mais iterações) com a intenção de se evitar a convergência rápida para ótimos locais de baixa qualidade.

A estratégia de seleção do primeiro aprimorante tenta evitar o custo de avaliar toda a vizinhança e seleciona o primeiro vizinho encontrado com melhoramento na função de avaliação. Formalmente, a cada solução s, o primeiro aprimorante é o primeiro s' pertencente a $N(s)$, tal que $g(s') < g(s)$. A ordem em que os vizinhos são avaliados pode ter impacto significativo no ótimo local encontrado pelo algoritmo. Uma busca local primeiro aprimorante com a ordenação fixa sempre vai encontrar o mesmo ótimo local. Já uma busca com a ordenação aleatória das soluções vizinhas pode retornar ótimos locais diferentes em cada execução do algoritmo. Assim, outra estratégia a ser relatada aqui é a escolha aleatória entre os aprimorantes. Essa estratégia é implementada, geralmente, com a estratégia primeiro aprimorante onde a busca na vizinhança é aleatorizada a cada iteração. Essa aleatorização pode ser feita com sorteio a cada iteração ou com um único sorteio no inicio do algoritmo onde um vetor com a ordem de escaneamento da vizinhança é definido e mantido até o final do processo. A estratégia de seleção do primeiro aprimorante com ordenação aleatória da busca na vizinhança tende a obter uma diversificação maior de soluções quando o processo é repetido. Um estudo de caso dessa estratégia será mostrado na próxima sessão quando tratarmos do algoritmo 2-OPT.

Um ótimo local em uma vizinhança pode não ser ótimo local em outra vizinhança, porém, o ótimo global será sempre um ótimo local independente da vizinhança escolhida. Outras formas de realizar a busca local foram elaboradas variando a função de vizinhança a cada iteração. Essas estrategias são chamadas de *Variable Neighborhood Descent (VND)* ou Descida em Vizinhança Variável e tem como intuito prolongar a busca local e obter ótimos locais de melhor qualidade. Uma forma clássica de se implementar um (VND) é mostrada pelo algoritmo 3.1. Onde dado uma solução corrente s a cada iteração seleciona-se o melhor vizinho s' da vizinhança k da solução s. Se $g(s') < g(s)$ a busca continua retornando a primeira vizinhança, caso contrário descarta-se s' e continua a busca na vizinhança $k+1$ [18].

Algoritmo 3.1: Descida em Vizinhança Variável (*Variable Neighborhood Descent, VND*)

```
1  VND(s,K_max){
2  enquanto Critério de parada faça
3  |  k = 1;
4  |  enquanto k < K_max faça
5  |  |  seleciona se o melhor s' ∈ N^(k)(s);
6  |  |  se g(s') ¡ g(s) então
7  |  |  |  s = s';
8  |  |  |  k = 1;
9  |  |  senão
10 |  |  |  k = k+1;
11 |  |  fim
12 |  fim
13 fim
14 retorna s;
15 }
```

O Algoritmo 3.1 mostra a implementação do *VND* usando a estratégia do melhor aprimorante, porém, também é comum utilizar a estratégia do primeiro aprimorante para reduzir o custo computacional de se verificar toda vizinhança. Com relação a estratégia de troca de vizinhança, ao invés de retornar para a primeira vizinhança após sair de um ótimo local como no algoritmo mostrado, existem também as seguintes estratégias:

- *Pipe VND (P-VND)*, que continua na vizinhança k após sair de um ótimo local, ou seja, a linha 8 do Algoritmo 3.1 seria excluída.
- *Cicle VND (C-VND)*, que segue para vizinhança $k+1$ após sair de um ótimo local, ou seja, a linha 8 do algoritmo 3.1 seria $k = k+1$.
- *Union VND (U-VND)*, busca um ótimo local em uma grande vizinhança gerada a partir da união das diferentes vizinhanças do algoritmo, como é feito em uma busca

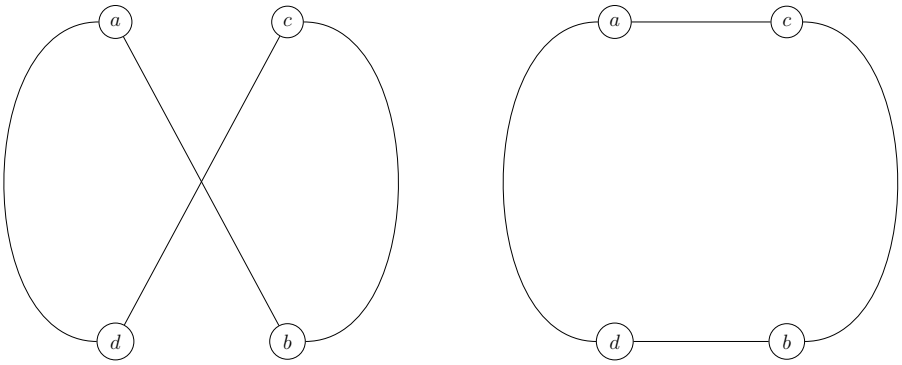

Figura 3.1: Movimento 2-OPT

local clássica. Tal abordagem é relatada nos trabalhos [57, 11, 45] que utilizam o *U-VND* como etapa de busca local inserida em outras meta-heurísticas.

Observe que, apesar do *VND* ser considerado uma variante do *VNS*, diferentemente do algoritmo *VNS* apresentado no capitulo anterior, o *VND* mostrado aqui não admite soluções que não sejam aprimorantes. Usando a troca de vizinhanças apenas para sair de ótimos locais e mantendo o gradiente de melhoria da solução corrente. Portanto, não foge do conceito de busca local, sendo que, no melhor caso, retorna o melhor ótimo local dentre todos ótimos locais de todas as vizinhanças. [18]

3.1 2-OPT

O algoritmo 2-OPT foi proposto pela primeira vez por [15]. Foi motivado pela seguinte observação para problemas euclidianos: se um ciclo Hamiltoniano cruza a si mesmo, ele pode ser facilmente reduzido, retirando as arestas que se cruzam e religando os dois caminhos desconexos com arestas que não se cruzam. Assim, o movimento básico do algoritmo 2-OPT consiste em remover duas arestas de um ciclo e reconectar os dois caminhos desconexos resultantes de uma forma diferente, resultando em um ciclo diferente do original. O movimento é descrito na Figura 3.1, em que obtemos uma nova solução, removendo as arestas (a, b) e (c, d), e religamos os caminhos com as arestas (a, c) e (b, d).

Definição 12 (Movimento 2-OPT). *Dado uma rota* $T = (v_1, ..., v_n, v_1)$ *e duas arestas* (v_i, v_{i+1}) *e* (v_j, v_{j+1}) *pertencentes a rota* T*. Definimos como movimento 2-OPT a operação de remoção das arestas* (v_i, v_{i+1}) *e* (v_j, v_{j+1}) *e adição das areatas* (v_i, v_j) *e* (v_{i+1}, v_{j+1}) *à rota* T*.*

Definição 13 (Vizinhança 2-OPT). *Dado uma rota $T = (v_1, ..., v_n, v_1)$ chamamos de vizinhança 2-OPT de T, todas as rotas geradas a partir de movimentos 2-OPT realizados em todos pares de arestas não consecutivas em T.*

O algoritmo 2-OPT pode ser definido da seguinte forma: O algoritmo inicia com uma rota qualquer T; Então, a função $N(T)$ retorna a vizinhança 2-OPT de T; Ao selecionar uma nova rota $T' \in N(T)$ melhor do que T, define-se uma nova vizinhança 2-OPT $N(T')$. Esse processo se repete sucessivamente até que não seja possível encontrar uma rota melhor que a corrente dentro da vizinhança 2-OPT.

[28] estudaram as diferenças entre as abordagens: Pior Aprimorante, Primeiro Aprimorante e Melhor Aprimorante aplicadas ao Caixeiro Viajante com a vizinhança 2-OPT. Segundo [28], se a solução inicial do processo for selecionada aleatoriamente, ao escolher o melhor aprimorante em cada iteração, temos resultados piores, na média, do que se selecionarmos o primeiro aprimorante. Contudo, se começar o processo a partir de uma solução inicial obtida através de alguma heurística construtiva, a abordagem que seleciona o Melhor Aprimorante se mostra melhor e mais rápida, na média, que a abordagem Primeiro Aprimorante em cada iteração. A abordagem Pior Aprimorante apresentou resultados piores que as outras abordagens em todos os casos. Segundo [28], a razão para esse comportamento seria que é melhor iniciar o processo fazendo as substituições em arestas menores, deixando as arestas maiores para o final do processo, quando serão mais fáceis de serem removidas.

[31] apresentaram um estudo de caso com a estrategia primeiro aprimorante com ordenação aleatória da vizinhança 2-OPT. O algoritmo usado foi elaborado da seguinte forma: A busca sempre começa a partir da mesma solução inicial, uma rota que visita os vértices do grafo em sua ordem canônica $(v_1, v_2, ..., v_n, v_1)$. Ao inicializar a busca, é gerada uma permutação aleatória dos números inteiros de 1 a n, o que determina a ordem em que a vizinhança será escaneada em cada etapa de busca. Essa permutação permanece inalterada durante todo o processo de busca. A busca é finalizada quando um ótimo local é encontrado para a vizinhança. Esse algoritmo foi executado 1000 vezes com a instância pcb3038 do TSPLIB. As soluções obtidas foram, em média, 8.6% a cima do ótimo global.

Vale ressaltar ainda o trabalho de [19], que demonstrou o limite superior no número iterações em uma busca local na vizinhança 2-OPT, tanto para soluções iniciais aleatórias quanto para soluções iniciais geradas por heurísticas construtivas. Por outro lado [38] estudaram a taxa de aproximação da abordagem 2-OPT e provaram um limite inferior para solução no pior caso.

3.2 1-Flip

Considerando o problema Max-Cut, a vizinhança de 1-flip consiste em todas as soluções que podem ser obtidas de uma solução movendo um vértice v do conjunto S para o conjunto $S\backslash V$ ou do conjunto $S\backslash V$ para o conjunto S. A diferença de custo entre uma solução e uma solução vizinha, obtida movendo um vértice v de um conjunto para o outro, é dada pela diferença entre o número de vértices adjacentes a v que estão no mesmo conjunto que v e o número de vértices adjacentes a v que estão no outro conjunto.

Definição 14 (Vizinhança 1-Flip). *Dado uma solução $S \subset V$, chamamos de vizinhança 1-Flip toda solução gerada a partir da troca de um vértice $v \in V$ de conjunto. Isto é, se $v \in S$ mude-o para o conjunto $S\backslash V$, $v \in S\backslash V$ mude-o para o conjunto S.*

3.3 Convergências Prematuras

Na Figura 3.2, temos um grafo hipotético gerado a partir de uma função de vizinhança aplicada a soluções de um problema de otimização combinatória. Nesse grafo direcionado os arcos ligam as soluções aos seus vizinhos aprimorantes. Como podemos ver, a solução inicial s1 tem como vizinhos aprimorantes s3, s4 e s5. Já a solução inicial s2 possui como aprimorante somente s7. Observe que todos os sumidouros desse grafo são ótimos locais, uma vez que não possuem nenhum vizinho aprimorante. Na figura 3.2 $g(s)$ representa a função objetivo do problema, portanto, quanto mais a baixo está o vértice melhor é a solução representada por este vértice.

Se uma busca local iniciada em s1 selecionar para a próxima iteração s3, terá como vizinho aprimorante na próxima iteração somente OL1, que é um Ótimo Local, ou, se selecionar s4 para a próxima iteração, terá como vizinhos aprimorantes OL1 e OL2, ambos Ótimos Locais. Porém, se selecionar s5, terá como vizinho aprimorante s6 que, por sua vez, terá como vizinho aprimorante OL3, que também é um Ótimo Local. Este último é uma solução melhor que OL1 ou OL2 de acordo com a função de avaliação $g(s)$. Portanto, definimos como sendo uma Convergência Prematura toda vez que uma busca local termina muito antes do que poderia, finalizando em um ótimo local de baixa qualidade se comparado a outros ótimos locais presentes em seu grafo de soluções aprimorantes.

Veja no grafo que, se iniciarmos a busca a partir de s2, teremos soluções diferentes do gerado a partir de s1, levando à solução OG, simbolizada aqui como Ótimo Global. Sendo assim, vemos que o grafo de soluções gerado a partir de soluções iniciais diferentes

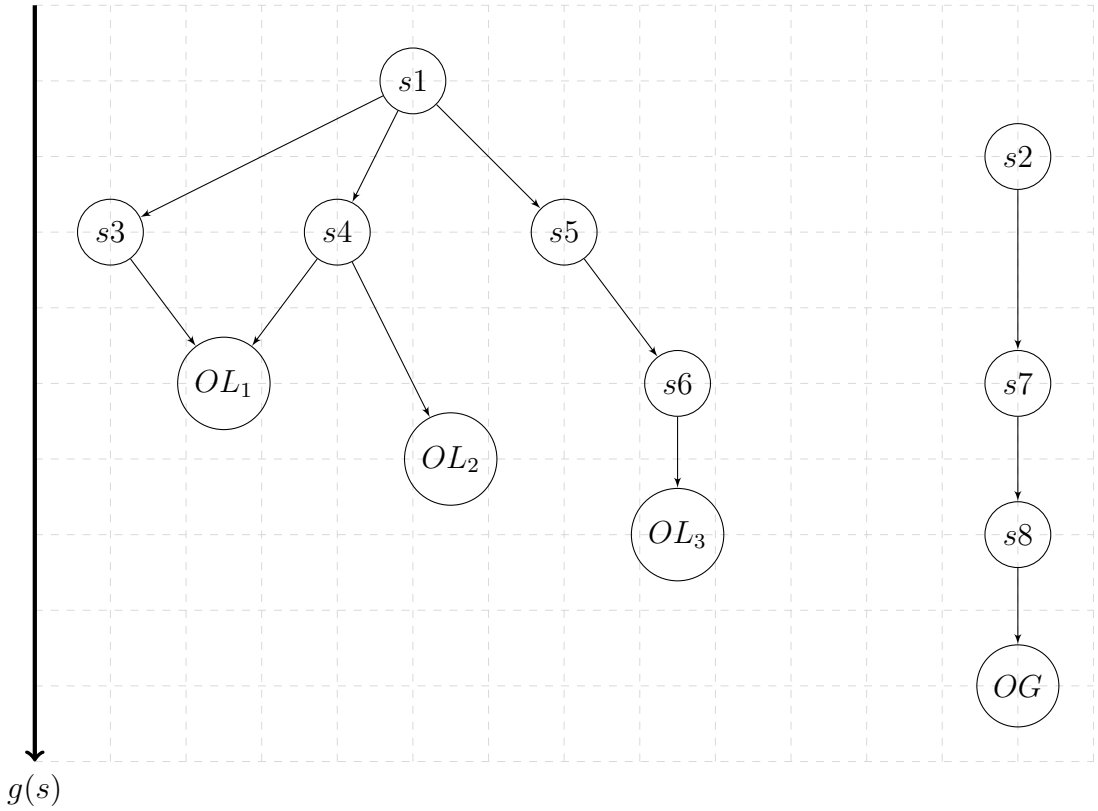

Figura 3.2: Representação simbólica de um grafo de vizinhança gerado no processo de Busca Local.

pode levar a soluções diferentes. Portanto a escolha da solução inicial influencia na Busca Local. O Ótimo Global pode não estar presente em um grafo de soluções partindo de uma solução específica e ser impossível de ser obtida partindo dessa solução.

3.4 Precondição de ótimos locais

[40] descrevem um método geral para formular cortes no conjunto de soluções viáveis de problemas de otimização combinatória. Os autores usam esses cortes na forma de desigualdades em formulações de programação linear inteira, que, por ser um método exato, tem por objetivo encontrar o ótimo global do problema. Tais cortes são motivados por algoritmos de buscas locais como o 2-OPT e separam o espaço de soluções viáveis entre soluções candidatas a ótimos locais e soluções impossíveis de serem ótimos locais. Os cortes reduzem o espaço de busca do algoritmo exato, uma vez que o ótimo global também é um ótimo local. Portanto limitar a busca entre soluções com características de ótimos locais aumentam as chances de encontrar o ótimo global mais rapidamente.

A busca local, por ser um procedimento heurístico, busca soluções suficientemente boas sem garantir que a solução encontrada seja um ótimo global. Dessa forma, retardar a convergência para um ótimo local de baixa qualidade, com intuito de obter melhores soluções, pode ser uma estratégia interessante em uma busca local. Então, decidimos usar as desigualdades propostas por [40] para selecionar a cada iteração as soluções aprimorantes com menos características de ótimos locais. A busca local proposta usa as desigualdades no sentido inverso ao usado no trabalho de [40] reduzindo desta forma as chances de uma convergência prematura. Tais desigualdades definem precondições para um ótimo local.

Descrevemos a seguir as desigualdades propostas por [40] baseadas na busca local 2-OPT. Dado um grafo $G(V, E)$, uma rota T em G e um subconjunto S com quatro vértices (sem perda de generalidade, assumimos $S = \{1, 2, 3, 4\}$) onde $S \subseteq V$. Particionamos $E(S) := \{(i, j) \in E : i, j \in S\}$ em três conjuntos disjuntos formados por duas arestas não adjacentes, como exemplificado na Figura 3.3, as arestas horizontais α, as arestas cruzando β e as arestas verticais γ . Sem perda de generalidade, assumimos $c(\alpha) \geq c(\beta) \geq c(\gamma)$ em que $c(p)$ é a soma dos custos das arestas formadas por cada par de vértices de $p \in \{\alpha, \beta, \gamma\}$. Considere a Fig. 3.3, onde $\alpha = \{(1, 2), (3, 4)\}$, $\beta = \{(1, 4), (2, 3)\}$ e $\gamma = \{(1, 3), (2, 4)\}$. Formulamos as igualdades, $x(\alpha) := x_{12} + x_{34}$, $x(\beta) := x_{14} + x_{23}$ e $x(\gamma) := x_{13} + x_{24}$, onde $x_{ij} = 1$ se a aresta (i, j) pertence a rota T e $x_{ij} = 0$, caso contrário.

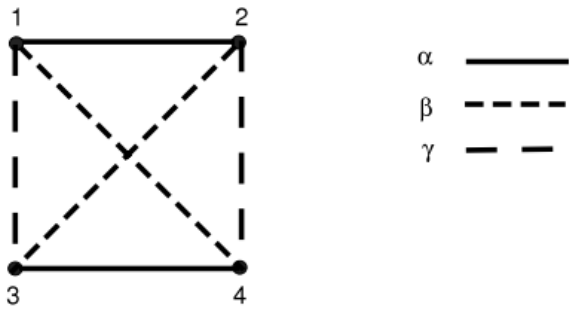

Figura 3.3: Subconjuntos de E (1 , 2 , 3 , 4), α , β e γ [40]

[40] fazem as seguintes afirmações sobre o uso da vizinhança 2-OPT para o Problema do Caxeiro Viajante.

Proposição 1. *Assumindo* $c(\alpha) > c(\beta) \geq c(\gamma)$.
Se a rota T *é um ótimo local então* $x(\alpha) \leq 1$.

Proposição 2. *Assumindo* $c(\alpha) \geq c(\beta) > c(\gamma)$.
Se a rota T *é um ótimo local então* $x(\alpha) + x(\beta) \leq 2$.

Proposição 3. *Assumindo,* $c(\alpha) > c(\beta) > c(\gamma)$.
Se a rota T *é um ótimo local então:*

(i) $2x(\alpha) + x(\beta) + x_{e^\gamma} \leq 3$ *para cada* $e^\gamma \in \gamma$

(ii) $2x(\alpha) + 2x_{e_1^\beta} + x_{e_2^\beta} + x(\gamma) \leq 4$ *para cada* $e_1^\beta, e_2^\beta \in \beta, e_1^\beta \neq e_2^\beta$

(iii) $3x(\alpha) + 2x(\beta) + x(\gamma) \leq 5$.

[40] faz ainda uma quarta proposição a respeito da vizinhança 2-OPT, porém, esta proposição não será usada no presente trabalho devido ao alto custo computacional para sua verificação.

Com relação ao problema do corte máximo (Max-Cut) [40] propõe as seguintes desigualdades. Considerando $x_{ij} = 1$ se somente se $|S \cap \{i, j\}| = 1$ para todo $i, j \in V$ temos:

Proposição 4. *Todo ótimo local deve satisfazer a seguinte desigualdade.*

$$\sum_{e \in \delta(i)} x_e \geq \left\lceil \frac{|\delta(i)|}{2} \right\rceil \; i \in V$$

Dado um conjunto de vértices V. Denotamos como $\delta(S)$ o conjunto de arestas em que uma terminação está em S e a outra está em $V \backslash S$. Com abuso de notação, usamos $\delta(v)$ invés de $\delta(\{v\})$ quando $|S| = 1$

3.5 *Delayed improvement local search (DILS)*

Inspirados nas proposições apresentadas foi elaborada a estratégia do aprimoramento adiado (*Delayed improvement local search (DILS)*), doravante denominado *DILS*, para seleção de soluções aprimorantes em buscas locais, a estratégia *DILS* considera as proposições apresentadas por [40], que são precondições para existência de ótimos locais, para selecionar as soluções viáveis que tenham menor possibilidade de serem ótimos locais, o algoritmo verifica se uma dada solução viola alguma das precondições apresentadas. Quanto maior for o número de precondições violadas por uma solução candidata, espera-se que mais distante estará de um ótimo local, ou seja, mais movimentos serão necessários para se tornar um ótimo local.

A estratégia *DILS* apresentada no algoritmo 3.2 pode ser descrita da seguinte forma: A busca tem início com uma solução viável s. Aplica-se a função de vizinhança $N(s)$. É selecionada dentre as soluções vizinhas, a solução aprimorante com maior número de violações das precondições s'. Define-se uma nova vizinhança $N(s')$. Esse processo se repete sucessivamente até que não seja possível encontrar uma solução melhor que a corrente dentro da vizinhança.

Algoritmo 3.2: Delayed improvement local search.

```
1 Delayed Improvement Local Search(s)
2 enquanto I(s) = {s' ∈ N(s) : f(s') < f(s)} não for vazio faça
3     k ← max_{s'∈I(s)} k_P(s');
4     s'' ← argmin_{s'∈I(s),k_P(s')=k} f(s');
5     s ← s'';
6 retorna s;
```

Foram elaborados os algoritmos de busca local 2-OPT e 1-Flip utilizando a estratégia *DILS*. O algoritmo 2-OPT utilizando a estratégia *DILS* será detalhado no Capitulo 4 onde também serão apresentados alguns resultados experimentais. O algoritmo 1-Flip utilizando a estratégia *DILS* será detalhado no Capitulo 6 onde também serão apresentados alguns resultados experimentais.

Capítulo 4

DILS aplicado ao 2-OPT

Neste capítulo, descreveremos a abordagem *DILS* aplicada ao algoritmo 2-OPT bem como os resultados obtidos.

Para selecionar as soluções viáveis que tenham menor possibilidade de conduzirem a ótimos locais rapidamente, o algoritmo verifica se uma dada solução viola alguma das precondições de existência de ótimos locais apresentadas por [40]. Quanto maior for o número precondições violadas por uma solução candidata, espera-se que mais distante esteja de um ótimo local, ou seja, mais movimentos serão necessários para se tornar um ótimo local. Contudo, apenas uma proposição violada já é o suficiente para uma solução candidata não ser um ótimo local como demonstrado por [40]. Esse processo é apresentado no Algoritmo 4.1 considerando as seguintes definições:

Definição 15 (**Número de precondições de ótimos locais violadas**). *Seja* $T = (v_1, v_2, ..., v_n, v_1)$ *um ciclo hamiltoniano em* G. *Denotamos* $P(T, S)$ *o número de precondições (precondições de ótimos locais) violadas pelos conjuntos* α, β *e* γ, *como demonstrado no capítulo anterior, formados exclusivamente por vértices do conjunto* $S \subseteq V$, *considerando para o cálculo as incidência das arestas de* T *nos vértices de* S, *tal que* $S = \{v_i, v_{i+1}, v_j, v_{j+1}\}$ *e* $i \neq j, i \neq j+1, j \neq i+1$ *para a rota* T. *Considerando* $S_1, S_2, ...S_k$ *todos os conjuntos* S *possíveis na rota* T, *o número de precondições violadas na rota* T *é dado por* $P(T) = \sum_{i=1}^{k} P(T, S_i)$.

Definição 16 (**Operação 2-OPT**). *Seja* $T' = opt2(T, i, j)$ *a operação de remoção das arestas* (v_i, v_{i+1}) *e* (v_j, v_{j+1}) *e adição das areatas* (v_i, v_j) *e* (v_{i+1}, v_{j+1}) *na rota* T.

Definição 17 (**Diferença de precondições violadas**). *Defina* $DifProp(T, T') = P(T') - P(T)$ *como a diferença entre o número de precondições violadas nas rotas* T *e* T'. *Perceba que, se* T' *for obtido a partir de uma operação 2-OPT em* T , *para avaliar a diferença de precondições violadas entre* T *e* T' *só será necessário considerar aqueles conjuntos* α, β *ou* γ *que contém, pelo menos, uma aresta escolhida para a executar a operação 2-OPT, pois o número de precondições violadas nos conjuntos que contém apenas arestas que não foram alteradas será exatamente o mesmo em* T *e* T'.

O algoritmo *DILS* inicia com uma solução inicial s. A cada iteração do algoritmo é selecionada dentre as soluções aprimorantes, a solução vizinha com maior número de

precondições violadas, ou seja, maior $DifProp(s, s')$. Como pode ser visto no Algoritmo 4.1. Na linha 4 a condição de parada do algoritmo é definida para quando não houver mais melhoras, a variável *melhorou* terá valor verdadeiro toda vez que for encontrado um vizinho aprimorante na vizinhança da iteração, ou seja, o algoritmo para quando na vizinhança da iteração corrente não houver nenhum vizinho aprimorante. Da linha 8 à linha 18 o algoritmo verifica toda a vizinhança $2 - OPT$ da solução corrente T. Nesse trecho, calcula-se o Δ_c e o Δ_p, que são a diferença entre o custo e as precondições violadas entre as rotas T e T' respectivamente. Na linha 12 se o Δ_p do vizinho avaliado for melhor que o melhor Δ_p já encontrado para essa vizinhança, esse vizinho será armazenado para para ser a próxima solução corrente ou até ser substituído por um vizinho melhor na mesma vizinhança. No caso em que houver um vizinho com Δ_p igual ao melhor Δ_p já encontrado, será armazenado para para ser a próxima solução corrente, o vizinho com Δ_c melhor que o melhor Δ_c já encontrado. Ao final da iteração a linha 19 verifica se foi encontrado um vizinho aprimorante, se sim, o transforma em solução corrente para

Algoritmo 4.1: *DILS* aplicado ao 2-OPT.

```
1  BuscaLocalDILS(T,G(V,E)){
2  Faça n = —T—.;
3  Faça melhorou = 1;
4  enquanto melhorou faça
5      Faça BestΔc = 0;
6      Faça BestΔp = 0;
7      Faça melhorou = 0;
8      para Cada par de vértices i, j ∈ V tal que i e j não sejam consecutivos em T
        faça
9          Δc = Ci,j + Ci+1,j+1 − Ci,i+1 − Cj,j+1;
10         T' = opt2(T,i,j);
11         Δp = DifProp(T, T');
12         se (Δp ¿ BestΔp e Δc ¡ 0) ou (Δp == BestΔp e Δc ¡ BestΔc ) então
13             i* = i;
14             j* = j;
15             BestΔp = Δp;
16             melhorou = 1;
17         fim
18     fim
19     se melhorou então
20         i = i*;
21         j = j*;
22         T = opt2(T,i,j);
23     fim
24 fim
25 retorna T;
26 }
```

próxima iteração. Caso contrário a busca termina e a linha 25 retorna o ótimo local encontrado.

Para avaliar o desempenho da abordagem apresentada realizamos experimentos executando a busca local 2-OPT com melhor aprimorante (do inglês *BILS*) e a abordagem *DILS* no mesmo conjunto de instâncias para o problema TSP. Os testes foram realizados em instâncias da biblioteca TSPLIB. Para cada instância, repetimos o experimento 100 vezes, com soluções iniciais geradas aleatoriamente e usada em ambas abordagens, i.e., as 100 execuções de cada abordagem usam as mesmas soluções iniciais geradas aleatoriamente. As máquinas utilizadas possuem sistema operacional Ubuntu 18.04.3 LTS, com 4 processadores 64bits Intel(R) Xeon(R) E5405, 2GHz e 16.0 GB de memória RAM. O conjunto de instâncias usado é apresentado na tabela 4.1 onde mostra o nome da instância, o número de vértices e o tipo de coordenadas da instância.

Tabela 4.1: Conjunto de instâncias da TSPLIB utilizada para avaliar a performance da abordagem DILS

Instância	Vértices	Tipo de Coordenadas
att48	48	Pseudo-euclidianas ATT
berlin52	52	Euclidianas 2D
burma14	14	Geográfica
kroA100	100	Euclidianas 2D
kroA150	150	Euclidianas 2D
kroB100	100	Euclidianas 2D
kroB150	150	Euclidianas 2D
kroC100	100	Euclidianas 2D
kroD100	100	Euclidianas 2D
kroE100	100	Euclidianas 2D
lin105	105	Euclidianas 2D
pr107	107	Euclidianas 2D
pr124	124	Euclidianas 2D

Na tabela 4.2 mostramos a sumarização dos resultados. Para cada instância apresentamos as médias de iterações nas abordagens 2-OPT com melhor aprimorante (*BILS*) e *DILS* e os custos da solução média obtida pelas abordagens *BILS* e *DILS*. Verificamos que a abordagem *DILS* obteve valores médios de ótimos locais melhores que a abordagem *BILS*.

Diante desses resultados e com o intuito de verificar a contribuição dada ao selecionar a solução com maior número precondições violadas nos resultados obtidos, executamos o algoritmos DILS com a seleção de solução aprimorante invertida, ou seja, selecionando a cada iteração a solução com menor número precondições violadas. Na tabela 4.3, são mostrados os resultados de 100 execuções do algoritmo com essa modificação. Nota-se que ao selecionar a solução aprimorante com a menor violação das precondições precisamos de um número maior de iterações na busca local em relação ao necessário na busca local

Tabela 4.2: Resultados para busca DILS e 2-OPT Melhor Aprimorante (BILS). Executando 100 vezes para cada instância a partir de soluções iniciais aleatórias.

Instância	Iterações		Custo	
	BILS	*DILS*	*BILS*	*DILS*
att48	43,76	573,04	11.059,31	10.922,93
berlin52	46,01	694,45	8.099,31	8.032,05
burma14	8,70	43,28	3.489,01	3.480,15
kroA100	105,45	2.491,80	22.766,53	22.081,71
kroA150	164,08	5.384,89	28.591,15	27.618,98
kroB150	163,72	5.231,39	28.100,13	27.440,53
kroC100	104,43	2.490,84	22.335,53	21.860,85
kroD100	103,53	2.458,25	22.804,79	22.249,23
kroE100	102,89	2.429,73	23.579,33	23.000,72
lin105	112,24	2.873,46	15.264,64	14.881,26
pr107	112,58	3.152,66	46.974,64	47.298,54
pr124	140,32	4.149,13	61.481,17	61.078,44

2-OPT-MA, porém, os ótimos locais obtidos são significativamente piores que os ótimos locais obtidos pela abordagem 2-OPT-MA.

Assim, apesar de precisar de mais passos até um ótimo local que a estratégia melhor aprimorante, a abordagem minimizando o número de precondições violadas não leva a soluções melhores. Para entender melhor esse processo plotamos as iterações da busca local para cada abordagem mostrando o número de precondições violadas na solução aprimorante selecionada a cada iteração. A Fig. 4.1 mostra o processo para um única execução com a instância bruma14 da TSPLIB, e a Fig. 4.2 mostra o processo para uma única execução com a instância kroA100 da TSPLIB. Nessas figuras observa-se que a abordagem que busca minimizar o número de precondições violadas tem um comportamento muito parecido com o 2-OPT clássico reduzindo abruptamente o número de precondições violadas a cada iteração. Já a abordagem DILS aumenta o número de precondições violadas nas primeiras iterações e depois reduz gradualmente a cada iteração, produzindo uma convergência mais gradual.

Executamos também testes comparando os resultado da abordagem DILS com a abordagem 2-OPT com primeiro aprimorante (2-OPT-PA). Os testes foram feitos nas mesmas condições dos testes executados ao comparar com a abordagem 2-OPT-MA. Os resultados são apresentados na Tabela 4.4. Observa-se que a abordagem DILS precisou de mais iterações que a abordagem 2-OPT-PA em todas as instancias. Obteve também, em média, ótimos locais melhores que os da abordagem 2-OPT-PA, com exceção da instancias pr107 onde a abordagem DILS foi significativamente pior. Observe que nesse caso todos os resultados dos testes foram significativos.

Na Tabela 4.5, mostramos, além da média, o melhor e pior ótimo local obtido para

Tabela 4.3: Resultados para busca DILS minimizando preposições violadas (DILSinv) e 2-OPT Melhor Aprimorante (BILS). Executando 100 vezes para cada instância a partir de soluções iniciais aleatórias.

Instância	Iterações			Custo		
	BILS	DILSinv	p-value	BILS	DILSinv.	p-value
att48	43,76	58,39	1,12E-17	11.059,31	11.415,76	3,00E-13
berlin52	46,01	58,94	5,05E-17	8.099,31	8.578,76	2,12E-15
burma14	8,70	10,15	1,15E-06	3.489,01	3.549,34	2,56E-11
kroA100	105,45	136,84	3,91E-18	22.766,53	24.454,31	1,63E-16
kroA150	164,08	206,24	4,02E-18	28.591,15	30.677,21	5,51E-18
kroB150	163,72	209,76	3,92E-18	28.100,13	30.007,14	5,19E-18
kroC100	104,43	136,41	4,16E-18	22.335,53	23.725,78	3,42E-16
kroD100	103,53	134,75	3,89E-18	22.804,79	24.219,49	8,39E-18
kroE100	102,89	130,37	4,14E-18	23.579,33	24.947,95	3,01E-17
lin105	112,24	144,44	3,93E-18	15.264,64	16.390,85	1,03E-17
pr107	112,58	162,43	3,92E-18	46.974,64	47.720,08	9,86E-05
pr124	140,32	174,77	3,92E-18	61.481,17	65.571,19	5,39E-16

100 execuções do algoritmo DILS, 2-OPT-MA e 2-OPT-PA. Mostramos o ótimo global conhecido na literatura para para todas as instâncias e o comparamos com o resultado de todas abordagens, mostramos o erro em relação à média e em relação ao melhor resultado. Os melhores resultados estão destacados em negrito. A abordagem DILS, além de obter soluções melhores na média dos resultados, como mostrado em negrito na Tabela 4.5, obtém, também, um número maior de soluções melhores, ou seja, valores mínimos absolutos, que as abordagens do 2-OPT-MA e 2-OPT-PA. Observa-se que média do erro em relação ao melhor resultado é melhor na a abordagem DILS. Nota-se também que as piores soluções obtidas pela abordagem DILS são melhores que as piores soluções obtidas pelas abordagens 2-OPT-MA e 2-OPT-PA.

Avaliamos também a performance da abordagem DILS a partir de soluções geradas por uma heurística construtiva. Realizamos experimentos executando a busca local DILS, 2-OPT-MA e 2-OPT-PA no mesmo conjunto de instância dos testes anteriores. As soluções iniciais foram geradas pela heurística de inserção do vizinho mais próximo mostrada no algoritmo 4.2. Essa heurística inicia a construção da rota a partir de um vértice aleatório, após isso, adiciona-se o vértice mais próximo do primeiro vértice adicionado. Então, sucessivamente, até não haver mais vértices a ser inserido, seleciona-se um vértice ainda não visitado e coloca-o na rota na posição onde a distância triangular para os vértices vizinhos é menor. Para cada instância, repetimos o experimento 100 vezes para cada abordagem usando as mesmas soluções iniciais geradas pela heurística construtiva.

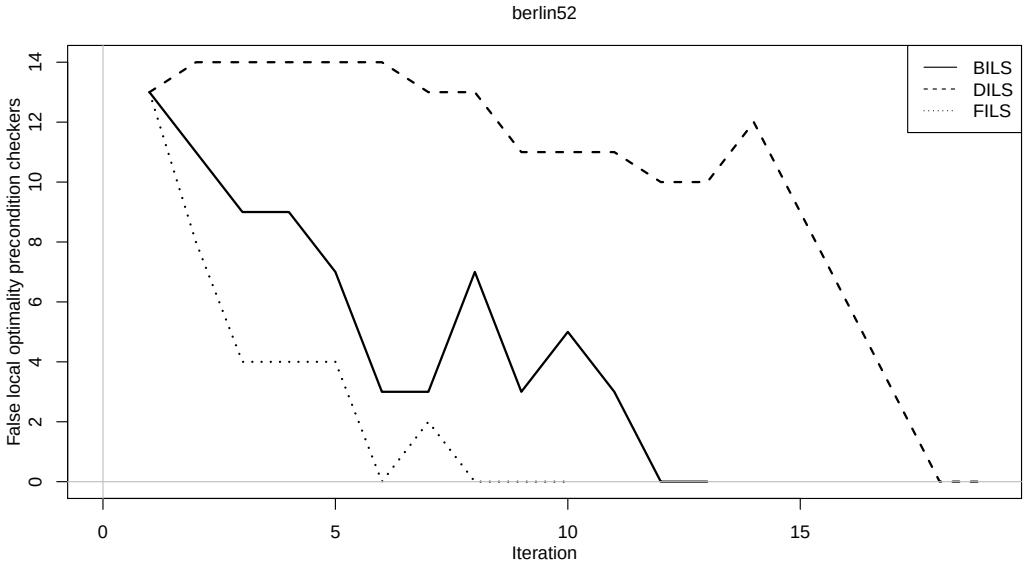

Figura 4.1: Passo a Passo para a instância berlin52

Foram utilizadas as mesmas máquinas dos testes anteriores.

Os resultados dos teste usando as soluções iniciais geradas por heurística construtivas são apresentados na tabela 4.6. Assim como observado quando usamos soluções inicias aleatórias, a abordagem DILS obteve soluções na média melhores e obteve a maioria das soluções melhores em relação aos resultados das abordagens do 2-OPT-MA e 2-OPT-PA, como pode ser visto pelos valores em negrito. Pode-se observar também que as piores soluções obtidas pela abordagem DILS são, na maioria dos casos, melhores que as piores soluções obtidas pelas abordagens 2-OPT-MA e 2-OPT-PA. Observa-se nas tabelas 4.5 e 4.6 que não é possível confirmar o que foi observado pelo trabalho [28].

Realizamos também testes com instâncias maiores, com mais de 150 vértices. Realizamos os experimentos executando a busca local DILS, 2-OPT-MA e 2-OPT-PA nas instâncias mostradas na tabela 4.7.

Os resultados são mostrados executando 1 vez para cada instância a a partir de uma solução inicial aleatória usada em todas as abordagens. Os resultados podem ser vistos na tabela 4.8. Veja que o tempo de execução para abordagem DILS é muito maior que o tempo de execução das outras abordagens. Tal resultado se deve ao número maior de iterações necessária para convergência do algoritmo DILS e ao fato de que calcular quantas precondições são violadas em uma rota é uma operação de ordem linear ($O(n)$). Já o cálculo do custo de uma troca 2-OPT tem custo constante C. Com relação ao número maior de iterações necessárias a para se convergir na abordagem DILS entende-se que é um custo necessário a se pagar ao se evitar cair em ótimos locais de baixa qualidade nas primeira iterações da busca. Mas uma análise precisa ser feita com relação ao custo de

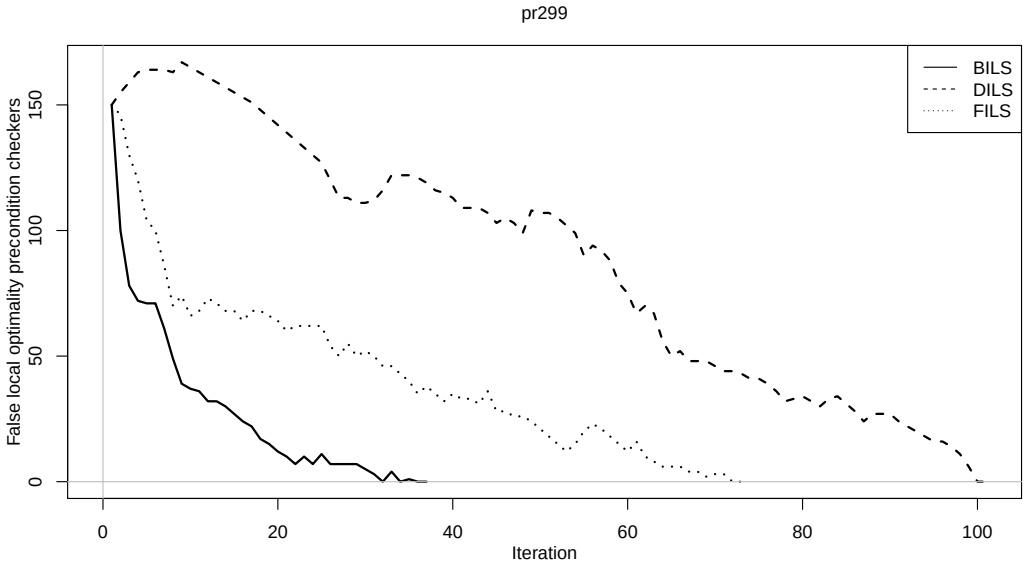

Figura 4.2: Passo a Passo para a instância pr299

Tabela 4.4: Resultados para busca DILS e 2-OPT Primeiro Aprimorante (FILS). Executando 100 vezes para cada instância a partir de soluções iniciais aleatórias.

Instância	Iterações			Custo		
	FILS	DILS	p-value	FILS	DILS	p-value
att48	143,91	573,04	3,95E-18	11.074,68	10.922,93	1,02E-07
berlin52	160,69	694,45	3,95E-18	8.164,02	8.032,05	1,59E-05
burma14	19,71	43,28	5,69E-18	3.507,85	3.480,15	5,78E-03
kroA100	395,16	2.491,80	3,95E-18	22.381,48	22.081,71	2,39E-04
kroA150	676,60	5.384,89	3,96E-18	28.298,71	27.618,98	4,92E-13
kroB150	675,79	5.231,39	3,95E-18	27.757,98	27.440,53	1,28E-05
kroC100	385,94	2.490,84	3,95E-18	22.113,19	21.860,85	4,20E-03
kroD100	392,67	2.458,25	3,95E-18	22.699,81	22.249,23	7,78E-10
kroE100	392,96	2.429,73	3,96E-18	23.364,29	23.000,72	2,50E-07
lin105	423,48	2.873,46	3,95E-18	15.296,52	14.881,26	2,02E-12
pr107	419,36	3.152,66	3,96E-18	47.049,40	47.298,54	4,27E-02
pr124	491,20	4.149,13	3,95E-18	61.599,79	61.078,44	8,01E-04

cada iteração que é alto em relação às abordagens 2-OPT clássicas. Diante disso, passou-se a trabalhar na redução do custo computacional do cálculo do número de precondições violadas. Esse processo é descrito no próximo capítulo onde descrevemos a segunda versão do algoritmo DILS.

Tabela 4.5: Detalhamento dos resultados das buscas DILS, 2-OPT Melhor Ap. e 2-OPT Primeiro Ap. Executando 100 vezes para cada instância a partir de soluções iniciais aleatórias.

Instância	2-OPT Melhor Ap.			Ótimo	E. Média	E. Melhor
	Média	Mín.	Máx.			
att48	11059,31	10736	12399	10628	4,06%	1,02%
berlin52	8099,31	**7542**	8756	7542	7,39%	0%
burma14	3489,01	**3454**	3634	3323	5%	3,94%
kroA100	22766,53	**21305**	24599	21282	6,98%	0,11%
kroA150	28591,15	27156	30385	26524	7,79%	2,38%
kroB150	28100,13	26828	30248	26130	7,54%	2,67%
kroC100	22335,53	**20819**	24005	20749	7,65%	0,34%
kroD100	22804,79	21834	24424	21294	7,09%	2,54%
kroE100	23579,33	22678	25450	22068	6,85%	2,76%
lin105	15264,64	14471	16359	14379	6,16%	0,64%
pr107	**46974,64**	**44833**	50626	44303	6,03%	1,20%
pr124	61481,17	**59223**	64801	59030	4,15%	0,33%
				Erro Médio	6,39%	1,49%
	2-OPT Primeiro Ap.					
	Média	Mín.	Máx.			
att48	11074,68	**10653**	11875	10628	4,20%	0,24%
berlin52	8164,02	7667	8734	7542	8,25%	1,66%
burma14	3507,85	**3454**	3759	3323	5,56%	3,94%
kroA100	22381,48	21485	24309	21282	5,17%	0,95%
kroA150	28298,71	27061	29812	26524	6,69%	2,02%
kroB150	27757,98	26642	29273	26130	6,23%	1,96%
kroC100	22113,19	20880	24362	20749	6,57%	0,63%
kroD100	22699,81	21519	23895	21294	6,60%	1,06%
kroE100	23364,29	22318	25508	22068	5,87%	1,13%
lin105	15296,52	14511	16771	14379	6,38%	0,92%
pr107	47049,4	45122	49753	44303	6,20%	1,85%
pr124	61599,79	59397	66982	59030	4,35%	0,62%
				Erro Médio	6,01%	1,42%
	DILS					
	Média	Mín.	Máx.			
att48	**10922,93**	10707	**11318**	10628	2,78%	0,74%
berlin52	**8032,05**	**7542**	**8386**	7542	6,50%	0%
burma14	**3480,15**	**3454**	**3519**	3323	4,73%	3,94%
kroA100	**22081,71**	21398	**22984**	21282	3,76%	0,55%
kroA150	**27618,98**	**26837**	**29034**	26524	4,13%	1,18%
kroB150	**27440,53**	**26634**	**28501**	26130	5,02%	1,93%
kroC100	**21860,85**	20969	**23128**	20749	5,36%	1,06%
kroD100	**22249,23**	**21518**	**23059**	21294	4,49%	1,05%
kroE100	**23000,72**	**22283**	**23850**	22068	4,23%	0,97%
lin105	**14881,26**	**14459**	**15610**	14379	3,49%	0,56%
pr107	47298,54	44951	**49030**	44303	6,76%	1,46%
pr124	**61078,44**	59412	**63459**	59030	3,47%	0,65%
				Erro Médio	4,56%	1,17%

Algoritmo 4.2: Algoritmo de Inserção do Vizinho Mais Próximo

```
1  InsercaoVMP(G(V,E)){
2  n = —V—;
3  Visitados = {};
4  T = {};
5  v_0 recebe um vértice aleatório de V;
6  Adicione v_0 a T;
7  Adicione v_0 a Visitados;
8  enquanto Número de cidade visitadas ¡ n faça
9      se Número de cidade visitadas = 1 então
10         v_prox = cidade mais próxima de v_0;
11         Adicione v_prox a T;
12         Adicione v_prox a Visitados;
13     senão
14         v_prox = qualquer vértice ainda não visitado de V;
15         p = posição com menor distancia triangular de v_prox em T;
16         Adicione v_prox na posição p em T;
17         Adicione v_prox a visitados;
18 fim
19 Retorne T;
20 }
```

Tabela 4.6: Detalhamento dos resultados das buscas DILS, 2-OPT Melhor Ap. e 2-OPT Primeiro Ap. Executando 100 vezes para cada instância a partir de soluções iniciais geradas com a heurística construtiva de inserção do vizinho mais próximo.

Instância	2-OPT Melhor Ap.			Ótimo	E. Média	E. Melhor
	Média	Mín.	Máx.			
att48	11,053.32	10,653	11,616	10,628	4.00%	0.24%
berlin52	8,176.69	7,604	**8,678**	7,542	8.42%	0.82%
burma14	3,493.04	**3,454**	3,634	3,323	5.12%	3.94%
kroA100	22,435.85	21,405	23,971	21,282	5.42%	0.58%
kroA150	28,276.88	27,484	29,614	26,524	6.61%	3.62%
kroB150	27,777.91	26,716	29,220	26,130	6.31%	2.24%
kroC100	22,019.92	**21,010**	23,669	20,749	6.13%	1.26%
kroD100	22,608.57	21,536	24,005	21,294	6.17%	1.14%
kroE100	23,448.80	22,417	**24,962**	22,068	6.26%	1.58%
lin105	15,400.92	14,567	16,320	14,379	7.11%	1.31%
pr107	45,987.28	**44,394**	49,118	44,303	3.80%	0.21%
pr124	61,430.67	59,087	**64,425**	59,030	4.07%	0.10%
				Erro Médio	5.78%	1.42%
	2-OPT Primeiro Ap.					
	Média	Mín.	Máx.			
att48	11,072.48	10,653	11,646	10,628	4.18%	0.24%
berlin52	8,239.75	7,700	8,831	7,542	9.25%	2.09%
burma14	3,502.88	**3,454**	3,634	3,323	5.41%	3.94%
kroA100	22,539.34	21,417	24,174	21,282	5.91%	0.63%
kroA150	28,427.98	27,349	29,716	26,524	7.18%	3.11%
kroB150	27,852.53	**26,468**	29,066	26,130	6.59%	1.29%
kroC100	22,165.95	21,240	**23,663**	20,749	6.83%	2.37%
kroD100	22,741.22	**21,529**	24,086	21,294	6.80%	1.10%
kroE100	23,546.50	22,541	25,015	22,068	6.70%	2.14%
lin105	15,529.66	14,589	16,438	14,379	8.00%	1.46%
pr107	46,202.70	44,566	48,253	44,303	4.29%	0.59%
pr124	61,704.94	**59,030**	65,161	59,030	4.53%	0.00%
				Erro Médio	6.31%	1.58%
	DILS					
	Média	Mín.	Máx.			
att48	**10,985.44**	**10,638**	**11,411**	10,628	3.36%	0.09%
berlin52	**8,143.73**	**7,542**	8,860	7,542	7.98%	0.00%
burma14	**3,487.03**	**3,454**	3,634	3,323	4.94%	3.94%
kroA100	**22,315.25**	**21,379**	**23,878**	21,282	4.86%	0.46%
kroA150	**28,117.87**	**27,262**	**29,612**	26,524	6.01%	2.78%
kroB150	**27,695.23**	26,490	**29,057**	26,130	5.99%	1.38%
kroC100	**21,974.99**	21,136	23,669	20,749	5.91%	1.87%
kroD100	**22,532.08**	21,632	**23,843**	21,294	5.81%	1.59%
kroE100	**23,266.86**	**22,335**	**24,962**	22,068	5.43%	1.21%
lin105	**15,304.80**	**14,548**	**16,144**	14,379	6.44%	1.18%
pr107	**45,921.84**	44,475	**48,153**	44,303	3.65%	0.39%
pr124	**61,345.95**	**59,030**	**64,425**	59,030	3.92%	0.00%
				Erro Médio	5.36%	1.24%

Tabela 4.7: Conjunto de instâncias da TSPLIB com mais de 150 vértices utilizado para avaliar a performance da abordagem DILS

Instância	Vértices	Tipo de Coordenadas
kroA200	200	Euclidianas 2D
kroB200	200	Euclidianas 2D
pr152	152	Euclidianas 2D
pr195	195	Euclidianas 2D
tsp225	225	Euclidianas 2D

Tabela 4.8: Detalhamento dos resultados das buscas DILS, 2-OPT Melhor Ap. e 2-OPT Primeiro Ap. Com instâncias maiores executando 1 vez para cada instância a partir de uma solução inicial aleatória.

Instância	2-OPT Melhor Ap.			2-OPT Primeiro Ap.			DILS		
	Custo	Tempo	Iter.	Custo	Tempo	Iter.s	Custo	Tempo	Iter.
kroA200	31216	0.040	224	30601	0.062	881	30393	813,688	7685
kroB200	32507	0.050	221	32012	0.041	991	31253	747,572	8388
pr152	78542	0,020	184	75152	0.019	685	79813	237,163	6303
rat195	2515	0,037	213	2477	0.037	797	2396	813,192	4999
tsp225	4411	0,098	244	4214	0.074	1078	4133	1131,684	7004

Capítulo 5

Aprimoramento da abordagem *DILS* aplicado ao 2-OPT

O algoritmo *DILS* aplicado ao 2-OPT, como foi projetado inicialmente, é muito lento em comparação ao 2-OPT clássico. Descrevemos nesse capítulo o trabalho realizado para aprimorar a abordagem *DILS* para o 2-OPT. A redução do tempo de execução foi feita analisando o impacto de cada movimento 2-OPT no número de precondições violadas na solução candidata. Criamos, assim, um algoritmo que calcula, em tempo constante, a diferença entre as precondições violadas para a maioria das soluções geradas na vizinhança.

5.1 Detalhes do Aprimoramento

A principal desvantagem do DILS é o esforço computacional associado à verificação das precondições otimalidade violadas para cada solução vizinha. A implementação ingênua de tal avaliação produziria um procedimento $O(n^4)$ para calcular o número de precondições de otimalidade violadas para cada vizinho.

Porém, inspecionando as precondições, pode-se verificar que todas elas, para serem avaliadas como falsas, precisam de pelo menos duas arestas não consecutivas envolvendo os nós a, b, c e d. Então, pode-se iterar sobre todos os pares de arestas da solução ao avaliar as peças. Assim, o tempo de cálculo necessário para a avaliação de todos as precondições em todos os pares possíveis é reduzido para $O(n^2)$.

Além disso, no contexto de uma pesquisa local, após a avaliação da primeira solução, não há necessidade de avaliar o número exato de precondições violadas, mas apenas a diferença entre o número de precondições violadas da solução atual e da solução vizinha que está sendo avaliada, chama aqui de Δ. No caso da vizinhança $2-opt$, apenas duas arestas são trocadas da solução atual para cada vizinha. Então, para calcular o Δ, é necessário avaliar a precondições apenas para os nós a, b, c e d que são os pontos finais das arestas que foram adicionadas ou removidas da solução atual.

No contexto de uma busca local $2-opt$, todos os verificadores podem ser avaliados para uma única vizinhança em etapas $O(n)$, uma vez que consideramos apenas os pares de arestas não consecutivas que estão presentes em uma solução e não na outra. Como apenas duas arestas são trocadas, o número de pares a serem considerados é limitado a $4n$.

Neste trabalho, melhoramos ainda mais, avaliando Δ para cada solução vizinha em tempo constante amortizado, resultando em tempo $O(n^2)$ para avaliar todas as $O(n^2)$ soluções na vizinhança. Para conseguir isso, calculamos e armazenamos dois valores para cada par de arestas não consecutivas na solução em cada iteração. O primeiro valor é Δ associado ao movimento 2-opt correspondente. O segundo, Δ', é o número de precondições violadas obtido ao realizar uma reconexão inválida dos caminhos após a remoção do par de arestas. Nesse caso, estamos salvando o número de precondições violadas para soluções inviáveis compostas por dois ciclos.

Com as informações de precondições violadas nas solução atual armazenada, podemos agora selecionar o movimento de aprimorante com maior Δ. Na próxima iteração os valores armazenados serão usados para atualizar os valores de Δ e Δ' de cada movimento.

A nova solução atual é muito semelhante à anterior. Difere na troca de um par de arestas. Sejam e e f as arestas que estavam presentes na solução anterior e não estão presentes na solução atual. Além disso, sejam g e h as arestas que substituíram as arestas e e f para obter a nova solução. Considere primeiro a atualização de Δ e Δ' para um par de arestas não consecutivas, sejam i e j, ambos diferentes das novas arestas g e h. Os valores de Δ e Δ' para esse par podem, de fato, mudar, pois é preciso calcular a diferença entre as precondições violadas no movimento anterior mas são não violadas no movimento atual, e vice-versa.

Isso significa que cálculos adicionais, para atualizar Δ e Δ', devem ser realizados para combinações de arestas envolvendo uma das arestas i ou j, bem como cada uma das quatro arestas g, h, e ou f. Como existem quatro arestas envolvidas no movimento avaliado (i, j e as arestas que os substituiriam) e quatro arestas que entraram ou saíram da solução na última iteração, há um total de 16 pares de arestas a serem considerados ao atualizar Δ, e 16 pares de arestas a serem considerados para atualizar Δ'. O número de pares é constante e, portanto, o custo de atualização Δ e Δ' também é constante.

Dependendo da atualização da solução na última iteração, o movimento envolvendo i e j pode agora precisar adicionar arestas diferentes do que antes para reconectar os dois caminhos que aparecem como resultado de sua remoção. Nesses casos, deve-se usar Δ' como base para atualizar Δ e Δ como base para atualizar Δ'. Como será exemplificado na Figura 5.2

Resta calcular Δ e Δ' para os movimentos envolvendo as novas arestas g e h introduzidas na solução na última iteração. Não temos nenhuma informação anterior sobre Δ e Δ' para esses movimentos, portanto, devemos calcular cada valor de Δ e Δ'

desde o início em tempo $O(n)$. Dado que existem $O(n)$ movimentos envolvendo as duas novas arestas, o tempo total para avaliar esses movimentos é $O(n^2)$.

Assim, temos $O(n^2)$ movimentos para os quais podemos avaliar Δ e Δ' em tempo constante e $O(n)$ movimentos que podem ser avaliados em tempo $O(n)$. A avaliação completa da vizinhança é $O(n^2)$, o que nos dá um tempo amortizado $O(1)$ para cada movimento.

5.2 Prova e Implementação do Aprimoramento

Como visto, calcular o número de precondições violadas no algoritmo *DILS* é uma operação de $O(n)$. Pois para cada vértices alterado na rota é necessário calcular o número de precondições violadas entre o novo vértice e cada vértice não consecutivo a ele, gerando, assim, $n-2$ cálculos par acada vértices alterado. O Teorema 1 prova que é possível fazer essa verificação em tempo menor em determinados casos.

Teorema 1. *Dado* $T = (v_1, v_2, ..., v_n, v_1)$ *uma rota em* $G(V, E)$, $T' = opt2(T, i, j)$, $T'' = opt2(T, k, l)$, $T''' = opt2(T', k, l)$, $\Delta_{p1} = DifProp(T, T')$, $\Delta_{p2} = DifProp(T, T'')$ *é possível calcular* $\Delta_{p3} = DifProp(T', T''')$ *em tempo constante, se somente se,* $V_1 \cap V_2 = \varnothing$ *ou* $V_1 \subset V_2$ *onde* $V_1 = \{v_i, ..., v_j\} \subseteq V$ *e* $V_2 = \{v_k, ..., v_l\} \subseteq V$ *com* $i > j$, $k > l$, $i \neq k$ *e* $j \neq l$.

Demonstração. Dado T uma rota em G, $T' = opt2(T, i, j)$, $T'' = opt2(T, k, l)$, $T''' = opt2(T', k, l)$, $\Delta_{p1} = DifProp(T, T')$, $\Delta_{p2} = DifProp(T, T'')$.

Note que as condições $V_1 \subset V_2$ e $V_1 \cap V_2 = \varnothing$ provocam o mesmo efeito na rota resultante da operação $2-OPT$. Pois, se tivermos $V_1 \subset V_2$, ao modificar o vértice inicial da rota T para algum vértice entre v_i e v_k, teremos $V_1 \cap V_2 = \varnothing$. Portanto, consideraremos nessa demonstração apenas a condição $V_1 \cap V_2 = \varnothing$.

Caso 1. Considerando $i < j, j < k, k < l, k - j > 1$ e $n - l + i > 1$. Temos:
$T = (v_1, v_2, ..., v_i, ..., v_j, ..., v_k, ..., v_l, ..., v_n, v_1)$
$T' = opt2(T, i, j) = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+1}, v_{j+1}, ..., v_k, ..., v_l, ..., v_n, v_1)$
$T'' = opt2(T, k, l) = (v_1, v_2, ..., v_i, ..., v_j, ..., v_k, v_l, v_{l-1}, ..., v_{k+1}, v_{l+1}, ..., v_n, v_1)$
$T''' = opt2(T', k, l) = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+1}, v_{j+1}, ..., v_k, v_l, ..., v_{l-1}, ..., v_{k+1},$
$k_{l+1}, ..., v_n, v_1)$

$\Delta_{p2} = DifProp(T, T'')$ será dado pela diferença das precondições violadas nos pares

de arestas não consecutivos em T e T'' afetados pela operação $opt2(T,k,l)$:

$$\Delta_{p2} = -P\left(T, \bigcup_{\substack{(v_r,v_s)\in T\\ \{v_r,v_s\}\cap\{v_k,v_{k+1}\}=\varnothing}} \{\{v_k,v_{k+1},v_r,v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T\\ \{v_r,v_s\}\cap\{v_l,v_{l+1}\}=\varnothing}} \{\{v_l,v_{l+1},v_r,v_s\}\}\right)$$

$$-P(T,\{\{v_{k-1},v_k,v_{k+1},v_{k+2}\},\{v_{l-1},v_l,v_{l+1},v_{l+2}\},\{v_{k-1},v_k,v_l,v_{l-1}\},\{v_{l+1},v_{l+2},v_{k+1},v_{k+2}\}\})$$

$$+P\left(T'', \bigcup_{\substack{(v_r,v_s)\in T''\\ \{v_r,v_s\}\cap\{v_k,v_l\}=\varnothing}} \{\{v_k,v_l,v_r,v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T''\\ \{v_r,v_s\}\cap\{v_{k+1},v_{l+1}\}=\varnothing}} \{\{v_{k+1},v_{l+1},v_r,v_s\}\}\right)$$

$$+P(T'',\{\{v_{k-1},v_k,v_l,v_{l-1}\},\{v_{k+2},v_{k+1},v_{l+1},v_{l+2}\},\{v_{k-1},v_k,v_{k+1},v_{k+2}\},\{v_{l-1},v_l,v_{l+1},v_{l+2}\}\}).$$

$\Delta_{p3} = DifProp(T',T''')$ será dado pela diferença das precondições violadas nos pares de arestas não consecutivos em T' e T''' afetados pela operação $opt2(T',k,l)$:

$$\Delta_{p3} = -P\left(T', \bigcup_{\substack{(v_r,v_s)\in T'\\ \{v_r,v_s\}\cap\{v_k,v_{k+1}\}=\varnothing}} \{\{v_k,v_{k+1},v_r,v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T'\\ \{v_r,v_s\}\cap\{v_l,v_{l+1}\}=\varnothing}} \{\{v_l,v_{l+1},v_r,v_s\}\}\right)$$

$$-P(T',\{\{v_{k-1},v_k,v_{k+1},v_{k+2}\},\{v_{l-1},v_l,v_{l+1},v_{l+2}\},\{v_{k-1},v_k,v_l,v_{l-1}\},\{v_{l+1},v_{l+2},v_{k+1},v_{k+2}\}\})$$

$$+P\left(T''', \bigcup_{\substack{(v_r,v_s)\in T'''\\ \{v_r,v_s\}\cap\{v_k,v_l\}=\varnothing}} \{\{v_k,v_l,v_r,v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T'''\\ \{v_r,v_s\}\cap\{v_{k+1},v_{l+1}\}=\varnothing}} \{\{v_{k+1},v_{l+1},v_r,v_s\}\}\right)$$

$$+P(T'',\{\{v_{k-1},v_k,v_l,v_{l-1}\},\{v_{k+2},v_{k+1},v_{l+1},v_{l+2}\},\{v_{k-1},v_k,v_{k+1},v_{k+2}\},\{v_{l-1},v_l,v_{l+1},v_{l+2}\}\}).$$

Observe que ao calcular Δ_{p3} verificamos conjuntos de vértices muito semelhantes aos verificados ao calcular Δ_{p2}, a diferença é que dentre as arestas $(v_r,v_s)\in T$ estão as arestas (v_i,v_{i+1}) e (v_j,v_{j+1}) que não fazem parte da rota T''. Como $T' = opt2(T,i,j)$, dentre as arestas $(v_r,v_s)\in T'$ existe as arestas (v_i,v_j) e (v_{i+1},v_{j+1}) que não existem na rota T. A mesma observação pode ser feita para as arestas $(v_r,v_s)\in T''$ e $(v_r,v_s)\in T'''$. Então, ao invés de recalcular todos pares de arestas não consecutivos em T' e T''' afetados pela operação $opt2(T',k,l)$, podemos usar o complemento dos conjuntos usados para calcular Δ_{p2} e Δ_{p3} para encontras as diferenças. Assim teremos:

$\Delta_{p3} = \Delta_{p2}$
$+(P(T,\{\{v_k,v_{k+1},v_i,v_{i+1}\},\{v_k,v_{k+1},v_j,v_{j+1}\},\{v_l,v_{l+1},v_i,v_{i+1}\},\{v_l,v_{l+1},v_j,v_{j+1}\}\}))$
$-(P(T',\{\{v_k,v_{k+1},v_i,v_j\},\{v_k,v_{k+1},v_{i+1},v_{j+1}\},\{v_l,v_{l+1},v_i,v_j\},\{v_l,v_{l+1},v_{i+1},v_{j+1}\}\}))$
$-(P(T'',\{\{v_k,v_l,v_j,v_{j+1}\},\{v_k,v_l,v_i,v_{i+1}\},\{v_{k+1},v_{l+1},v_j,v_{j+1}\},\{v_{k+1},v_{l+1},v_i,v_{i+1}\}\}))$
$+(P(T''',\{\{v_k,v_l,v_{i+1},v_{j+1}\},\{v_k,v_l,v_i,v_j\},\{v_{k+1},v_{l+1},v_{i+1},v_{j+1}\},\{v_{k+1},v_{l+1},v_i,v_j\}\})).$

Como número de conjuntos a serem calculados é constante, é possível calcular Δ_{p3} em tempo constante no Caso 1.

Caso 2. Considerando $i < j, j < k, k < l, k - j = 1$ e $n - l + i > 1$. Temos:
$T = (v_1, v_2, ..., v_i, ..., v_j, v_k, ..., v_l, ..., v_n, v_1)$
$T' = opt2(T, i, j) = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+1}, v_k, ..., v_l, ..., v_n, v_1)$
$T'' = opt2(T, k, l) = (v_1, v_2, ..., v_i, ..., v_j, v_k, v_l, v_{l-1}, ..., v_{k+1}, v_{l+1}, ..., v_n, v_1)$
$T''' = opt2(T', k, l) = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+1}, v_k, v_l, v_{l-1}, ..., v_{k_1}$
$, v_{l+1}, ..., v_n, v_1)$

$\Delta_{p2} = DifProp(T, T'')$ será dado pela diferença das precondições violadas nos pares de arestas não consecutivos em T e T'' afetados pela operação $opt2(T, k, l)$:

$$\Delta_{p2} = -P\left(T, \bigcup_{\substack{(v_r,v_s) \in T \\ \{v_r,v_s\}\cap\{v_k,v_{k+1}\}=\varnothing}} \{\{v_k, v_{k+1}, v_r, v_s\}\} \cup \bigcup_{\substack{(v_r,v_s) \in T \\ \{v_r,v_s\}\cap\{v_l,v_{l+1}\}=\varnothing}} \{\{v_l, v_{l+1}, v_r, v_s\}\}\right)$$

$$-P(T, \{\{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}, \{v_{l-1}, v_l, v_{l+1}, v_{l+2}\}, \{v_{k-1}, v_k, v_l, v_{l-1}\}, \{v_{k+2}, v_{k+1}, v_{l+1}, v_{l+2}\}\})$$

$$+P\left(T'', \bigcup_{\substack{(v_r,v_s) \in T'' \\ \{v_r,v_s\}\cap\{v_k,v_l\}=\varnothing}} \{\{v_k, v_l, v_r, v_s\}\} \cup \bigcup_{\substack{(v_r,v_s) \in T'' \\ \{v_r,v_s\}\cap\{v_{k+1},v_{l+1}\}=\varnothing}} \{\{v_{k+1}, v_{l+1}, v_r, v_s\}\}\right)$$

$$+P(T'', \{\{v_{k-1}, v_k, v_l, v_{l-1}\}, \{v_{k+2}, v_{k+1}, v_{l+1}, v_{l+2}\}, \{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}, \{v_{l-1}, v_l, v_{l+1}, v_{l+2}\}\}).$$

$\Delta_{p3} = DifProp(T', T''')$ será dado pela diferença das precondições violadas nos pares de arestas não consecutivos em T' e T''' afetados pela operação $opt2(T', k, l)$:

$$\Delta_{p3} = -P\left(T', \bigcup_{\substack{(v_r,v_s) \in T' \\ \{v_r,v_s\}\cap\{v_k,v_{k+1}\}=\varnothing}} \{\{v_k, v_{k+1}, v_r, v_s\}\} \cup \bigcup_{\substack{(v_r,v_s) \in T' \\ \{v_r,v_s\}\cap\{v_l,v_{l+1}\}=\varnothing}} \{\{v_l, v_{l+1}, v_r, v_s\}\}\right)$$

$$-P(T', \{\{v_{i+1}, v_k, v_{k+1}, v_{k+2}\}, \{v_{l-1}, v_l, v_{l+1}, v_{l+2}\}, \{v_{i+1}, v_k, v_{l-1}, v_l\}, \{v_{k+1}, v_{k+2}, v_{l+1}, v_{l+2}\}\})$$

$$+P\left(T''', \bigcup_{\substack{(v_r,v_s) \in T''' \\ \{v_r,v_s\}\cap\{v_k,v_l\}=\varnothing}} \{\{v_k, v_l, v_r, v_s\}\} \cup \bigcup_{\substack{(v_r,v_s) \in T''' \\ \{v_r,v_s\}\cap\{v_{k+1},v_{l+1}\}=\varnothing}} \{\{v_{k+1}, v_{l+1}, v_r, v_s\}\}\right)$$

$$+P(T''', \{\{v_{i+1}, v_k, v_l, v_{l-1}\}, \{v_{k+2}, v_{k+1}, v_{l+1}, v_{l+2}\}, \{v_{i+1}, v_k, v_{k+2}, v_{k+1}\}, \{v_l, v_{l-1}, v_{l+1}, v_{l+2}\}\}).$$

Observe que, assim como no Caso 1, ao calcular Δ_{p3} verificamos conjuntos de vértices muito semelhantes aos verificados ao calcular Δ_{p2}.Porém, no Caso 2 teremos as seguintes diferenças em relação ao Caso 1:

- Os conjuntos de vértices formados por pelas arestas (v_k, v_{k+1}) e (v_j, v_{j+1}) não serão calculados para rota T pois o vértice $v_{j+1} = v_k$ e, portanto, $\{v_j, v_{j+1}\} \cap \{v_k, v_{k+1}\} \neq \varnothing$.

- Os conjuntos de vértices formados por pelas arestas (v_k, v_{k+1}) e (v_{i+1}, v_{j+1}) não serão

calculados para rota T' pois o vértice $v_{j+1} = v_k$ e, portanto, $\{v_{i+1}, v_{j+1}\} \cap \{v_k, v_{k+1}\} \neq \varnothing$. O conjunto formado pelas arestas vizinhas a (v_k, v_{k+1}), que em T' é (v_{i+1}, v_k), (v_{k+1}, v_{k+2}), será calculado diferentemente da forma que foi calculado na rota T que foi (v_{k-1}, v_k), (v_{k+1}, v_{k+2}).

- Os conjuntos de vértices formados por pelas arestas (v_k, v_l) e (v_j, v_{j+1}) não serão calculados para rota T'' pois o vértice $v_{j+1} = v_k$ e, portanto, $\{v_j, v_{j+1}\} \cap \{v_k, v_l\} \neq \varnothing$.
- Os conjuntos de vértices formados por pelas arestas (v_k, v_l) e (v_{i+1}, v_{j+1}) não serão calculados para rota T''' pois o vértice $v_{j+1} = v_k$ e, portanto, $\{v_{i+1}, v_{j+1}\} \cap \{v_k, v_l\} \neq \varnothing$. O conjunto formado pelas arestas vizinhas a (v_k, v_l), que em T''' é (v_{i+1}, v_k), (v_l, v_{l-1}), será calculado diferentemente da forma que foi calculado na rota T'' que foi (v_{k-1}, v_k), (v_l, v_{l-1}).

Assim, teremos:

$\Delta_{p3} = \Delta_{p2}$
$+(P(T, \{\{v_k, v_{k+1}, v_i, v_{i+1}\}, \{v_l, v_{l+1}, v_i, v_{i+1}\}, \{v_l, v_{l+1}, v_j, v_{j+1}\}, \{v_{k-1}, v_k, v_l, v_{l-1}\},$
$\{v_{k-1}, v_k, v_l, v_{l-1}\}\}))$
$-(P(T', \{\{v_k, v_{k+1}, v_i, v_j\}, \{v_l, v_{l+1}, v_i, v_j\}, \{v_l, v_{l+1}, v_{i+1}, v_{j+1}\}, \{v_{i+1}, v_k, v_{k+1}, v_{k+2}\},$
$\{v_{i+1}, v_k, v_{l-1}, v_l\}\}))$
$-(P(T'', \{\{v_k, v_l, v_i, v_{i+1}\}, \{v_{k+1}, v_{l+1}, v_j, v_{j+1}\}, \{v_{k+1}, v_{l+1}, v_i, v_{i+1}\}, \{v_{k-1}, v_k, v_l, v_{l-1}\},$
$\{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}\}))$
$+(P(T''', \{\{v_k, v_l, v_i, v_j\}, \{v_{k+1}, v_{l+1}, v_{i+1}, v_{j+1}\}, \{v_{k+1}, v_{l+1}, v_i, v_j\}, \{v_{i+1}, v_k, v_l, v_{l-1}\}$
$\{v_{i+1}, v_k, v_{k+2}, v_{k+1}\}\})).$

Como o número de conjuntos a ser calculado é constante, é possível calcular Δ_{p3} em tempo constante no Caso 2.

Caso 3. Considerando $i < j, j < k, k < l, k - j > 1$ e $n - l + i = 1$. Temos:
$T = (v_i, ..., v_j, ..., v_k, ..., k_l, v_i)$

Basta alterarmos o vértice inicial da rota para qualquer vértices entre v_j e v_k teremos um caso análogo ao caso 2.

Caso 4. Considerando $i < j, j < k, k < l, k - j = 1$ e $n - l + i = 1$. Temos:
$T = (v_i, ..., v_j, v_k, ..., k_l, v_i)$
$T' = opt2(T, i, j) = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+1}, v_k, ..., v_l, ..., v_n, v_1)$
$T'' = opt2(T, k, l) = (v_1, v_2, ..., v_i, ..., v_j, v_k, v_l, v_{l-1}, ..., v_{k+1}, v_{l+1}, ..., v_n, v_1)$
$T''' = opt2(T', k, l) = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{i+1}, v_k, v_l, v_{l-1}, ..., v_{k_1}$
$, v_{l+1}, ..., v_n, v_1)$

$\Delta_{p2} = DifProp(T, T'')$ será dado pela diferença das precondições violadas nos pares

de arestas não consecutivos em T e T'' afetados pela operação $opt2(T,k,l)$:

$$\Delta_{p2} = -P\left(T, \bigcup_{\substack{(v_r,v_s)\in T\\ \{v_r,v_s\}\cap\{v_k,v_{k+1}\}=\varnothing}} \{\{v_k,v_{k+1},v_r,v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T\\ \{v_r,v_s\}\cap\{v_l,v_{l+1}\}=\varnothing}} \{\{v_l,v_{l+1},v_r,v_s\}\}\right)$$

$$-P(T,\{\{v_{k-1},v_k,v_{k+1},v_{k+2}\},\{v_{l-1},v_l,v_{l+1},v_{l+2}\},\{v_{k-1},v_k,v_l,v_{l-1}\},\{v_{k+2},v_{k+1},v_{l+1},v_{l+2}\}\})$$

$$+P\left(T'', \bigcup_{\substack{(v_r,v_s)\in T''\\ \{v_r,v_s\}\cap\{v_k,v_l\}=\varnothing}} \{\{v_k,v_l,v_r,v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T''\\ \{v_r,v_s\}\cap\{v_{k+1},v_{l+1}\}=\varnothing}} \{\{v_{k+1},v_{l+1},v_r,v_s\}\}\right)$$

$$+P(T'',\{\{v_{k-1},v_k,v_l,v_{l-1}\},\{v_{k+2},v_{k+1},v_{l+1},v_{l+2}\},\{v_{k-1},v_k,v_{k+1},v_{k+2}\},\{v_{l-1},v_l,v_{l+1},v_{l+2}\}\}).$$

$\Delta_{p3} = DifProp(T',T''')$ será dado pela diferença das precondições violadas nos pares de arestas não consecutivos em T' e T''' afetados pela operação $opt2(T',k,l)$:

$$\Delta_{p3} = -P\left(T', \bigcup_{\substack{(v_r,v_s)\in T'\\ \{v_r,v_s\}\cap\{v_k,v_{k+1}\}=\varnothing}} \{\{v_k,v_{k+1},v_r,v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T'\\ \{v_r,v_s\}\cap\{v_l,v_{l+1}\}=\varnothing}} \{\{v_l,v_{l+1},v_r,v_s\}\}\right)$$

$$-P(T',\{\{v_{i+1},v_k,v_{k+1},v_{k+2}\},\{v_{l-1},v_l,v_{l+1},v_{l+2}\},\{v_{i+1},v_k,v_{l-1},v_l\},\{v_{k+1},v_{k+2},v_{l+1},v_{l+2}\}\})$$

$$+P\left(T''', \bigcup_{\substack{(v_r,v_s)\in T'''\\ \{v_r,v_s\}\cap\{v_k,v_l\}=\varnothing}} \{\{v_k,v_l,v_r,v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T'''\\ \{v_r,v_s\}\cap\{v_{k+1},v_{l+1}\}=\varnothing}} \{\{v_{k+1},v_{l+1},v_r,v_s\}\}\right)$$

$$+P(T''',\{\{v_{i+1},v_k,v_l,v_{l-1}\},\{v_{k+2},v_{k+1},v_{l+1},v_{l+2}\},\{v_{i+1},v_k,v_{k+2},v_{k+1}\},\{v_l,v_{l-1},v_{l+1},v_{l+2}\}\}).$$

Observe que, assim como no Caso 1 e Caso 2, ao calcular Δ_{p3}, verificamos conjuntos de vértices muito semelhantes aos verificados ao calcular Δ_{p2}. Porém, no Caso 3 teremos as seguintes diferenças em relação ao Caso 1:

- Os conjuntos de vértices formados por pelas arestas $((v_k,v_{k+1}),(v_j,v_{j+1}))$ e $((v_l,v_{l+1}),(v_i,v_{i+1}))$ não serão calculados para rota T pois o vértice $v_{j+1} = v_k$ e o vértice $v_{l+1} = v_i$, portanto, $\{v_j,v_{j+1}\}\cap\{v_k,v_{k+1}\} \neq \varnothing$ e $\{v_i,v_{i+1}\}\cap\{v_l,v_{l+1}\} \neq \varnothing$.

- Os conjuntos de vértices formados por pelas arestas $((v_k,v_{k+1}),(v_{i+1},v_{j+1}))$ e $((v_l,v_{l+1}),(v_i,v_j))$ não serão calculados para rota T' pois o vértice $v_{j+1} = v_k$ e o vértice $v_{l+1} = v_i$, portanto, $\{v_{i+1},v_{j+1}\}\cap\{v_k,v_{k+1}\} \neq \varnothing$ e $\{v_i,v_j\}\cap\{v_l,v_{l+1}\} \neq \varnothing$.Os conjuntos formados pelas arestas vizinhas a (v_k,v_{k+1}) e (v_l,v_{l+1}), que em T' são (v_{i+1},v_k), (v_{k+1},v_{k+2}), (v_{l-1},v_l), (v_{l+1},v_j) será calculado diferentemente da forma que foi calculado na rota T que foi (v_{k-1},v_k), (v_{k+1},v_{k+2}),(v_{l-1},v_l), (v_{l+1},v_{l+2}).

- Os conjuntos de vértices formados por pelas arestas $((v_k,v_l),(v_j,v_{j+1}))$ e $((v_{k+1},v_{l+1}),(v_i,v_{i+1}))$ não serão calculados para rota T'' pois o vértice $v_{j+1} = v_k$ e o vértice $v_{l+1} = v_i$, portanto, $\{v_j,v_{j+1}\}\cap\{v_k,v_l\} \neq \varnothing$ e $\{v_i,v_{i+1}\}\cap\{v_{k+1},v_{l+1}\} \neq \varnothing$.

- Os conjuntos de vértices formados por pelos pares de arestas $((v_k, v_l), (v_{i+1}, v_{j+1}))$ e $((v_{k+1}, v_{l+1}), (v_i, v_j))$ não serão calculados para rota T''' pois o vértice $v_{j+1} = v_k$ e o vértice $v_{i+1} = v_l$, portanto, $\{v_{i+1}, v_{j+1}\} \cap \{v_k, v_l\} \neq \varnothing$ e $\{v_i, v_j\} \cap \{v_{k+1}, v_{l+1}\} \neq \varnothing$.O conjunto formado pelas arestas vizinhas a (v_k, v_l), que em T''' é (v_{i+1}, v_k), (v_l, v_{l-1}), será calculado diferentemente da forma que foi calculado na rota T'' que foi (v_{k-1}, v_k), (v_l, v_{l-1}).

Assim teremos:

$\Delta_{p3} = \Delta_{p2}$
$+(P(T, \{\{v_k, v_{k+1}, v_i, v_{i+1}\}, \{v_l, v_{l+1}, v_j, v_{j+1}\}, \{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}, \{v_{l-1}, v_l, v_{l+1}, v_{l+2}\},$
$\{v_{k-1}, v_k, v_l, v_{l-1}\}, \{v_{k+2}, v_{k+1}, v_{l+1}, v_{l+2}\}\}))$
$-(P(T', \{\{v_k, v_{k+1}, v_i, v_j\}, \{v_l, v_{l+1}, v_{i+1}, v_{j+1}\}, \{v_{k-1}, v_k, v_l, v_{l-1}\}, \{v_{k+2}, v_{k+1}, v_{l+1}, v_{l+2}\},$
$\{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}, \{v_{l-1}, v_l, v_{l+1}, v_{l+2}\}\}))$
$-(P(T'', \{\{v_k, v_l, v_i, v_{i+1}\}, \{v_{k+1}, v_{l+1}, v_j, v_{j+1}\}, \{v_{i+1}, v_k, v_{k+1}, v_{k+2}\}, \{v_{l-1}, v_l, v_{l+1}, v_{l+2}\},$
$\{v_{i+1}, v_k, v_{l-1}, v_l\}, \{v_{k+1}, v_{k+2}, v_{l+1}, v_{l+2}\}\}))$
$+(P(T''', \{\{v_k, v_l, v_i, v_j\}, \{v_{k+1}, v_{l+1}, v_{i+1}, v_{j+1}\}, \{v_{i+1}, v_k, v_l, v_{l-1}\}, \{v_{k+2}, v_{k+1}, v_{l+1}, v_{l+2}\},$
$\{v_{i+1}, v_k, v_{k+2}, v_{k+1}\}, \{v_l, v_{l-1}, v_{l+1}, v_{l+2}\}\})).$

Como o número de conjuntos a ser calculado é constante, é possível calcular Δ_{p3} em tempo constante no Caso 4.

Por outro lado, se consideramos $V_1 \cap V_2 \neq \varnothing$ e $V_1 \not\subset V_2$, temos:
$T = (v_1, v_2, ..., v_i, ..., v_k, v_{k+1}, ..., v_j, v_{j+1}, ..., v_l, ..., v_n, v_1)$
$T' = opt2(T, i, j) = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., , v_{k+1}, v_k, v_{k-1}..., v_{i+1}, v_{j+1}, ..., v_l, ..., v_n, v_1)$
$T'' = opt2(T, k, l) = (v_1, v_2, ..., v_i, ..., v_k, v_l, v_{l-1}, ..., v_{j+1}, v_j, ...v_{k+1}, v_{l+1}, ..., v_n, v_1)$
$T''' = opt2(T', k, l) = (v_1, v_2, ..., v_i, v_j, v_{j-1}, ..., v_{k+1}, v_k, v_l, ..., v_{j+1}, v_{i+1}, ..., v_{k-1}, v_{l+1}, ..., v_n, v_1)$

$\Delta_{p2} = DifProp(T, T'')$ será dado pela diferença das precondições violadas nos pares de arestas não consecutivos em T e T'' afetados pela operação $opt2(T, k, l)$:

$$\Delta_{p2} = -P\left(T, \bigcup_{\substack{(v_r,v_s) \in T \\ \{v_r,v_s\} \cap \{v_k,v_{k+1}\} = \varnothing}} \{\{v_k, v_{k+1}, v_r, v_s\}\} \cup \bigcup_{\substack{(v_r,v_s) \in T \\ \{v_r,v_s\} \cap \{v_l,v_{l+1}\} = \varnothing}} \{\{v_l, v_{l+1}, v_r, v_s\}\}\right)$$

$$-P(T, \{\{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}, \{v_{l-1}, v_l, v_{l+1}, v_{l+2}\}, \{v_{k-1}, v_k, v_l, v_{l-1}\}, \{v_{k+2}, v_{k+1}, v_{l+1}, v_{l+2}\}\})$$

$$+P\left(T'', \bigcup_{\substack{(v_r,v_s) \in T'' \\ \{v_r,v_s\} \cap \{v_k,v_l\} = \varnothing}} \{\{v_k, v_l, v_r, v_s\}\} \cup \bigcup_{\substack{(v_r,v_s) \in T'' \\ \{v_r,v_s\} \cap \{v_{k+1},v_{l+1}\} = \varnothing}} \{\{v_{k+1}, v_{l+1}, v_r, v_s\}\}\right)$$

$$+P(T'', \{\{v_{k-1}, v_k, v_l, v_{l-1}\}, \{v_{k+2}, v_{k+1}, v_{l+1}, v_{l+2}\}, \{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}, \{v_{l-1}, v_l, v_{l+1}, v_{l+2}\}\}).$$

Observe que em T' o sentido da arestas (v_k, v_{k+1}) ficou invertido fazendo com que

a operação $T''' = opt2(T', k, l)$ adicionasse a aresta (v_{k-1}, v_{l+1}) à rota T'''. Dessa forma $\Delta_{p3} = DifProp(T', T''')$, que é dado pela diferença das precondições violadas nos pares de arestas não consecutivos em T' e T''' afetados pela operação $opt2(T', k, l)$, será:

$$\Delta_{p3} = -P\left(T', \bigcup_{\substack{(v_r,v_s)\in T' \\ \{v_r,v_s\}\cap\{v_k,v_{k+1}\}=\varnothing}} \{\{v_k, v_{k+1}, v_r, v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T' \\ \{v_r,v_s\}\cap\{v_l,v_{l+1}\}=\varnothing}} \{\{v_l, v_{l+1}, v_r, v_s\}\}\right)$$

$$-P(T', \{\{v_{k-1}, v_k, v_{k+1}, v_{k+2}\}, \{v_{l-1}, v_l, v_{l+1}, v_{l+2}\}, \{v_{k-1}, v_k, v_{l-1}, v_l\}, \{v_{k+1}, v_{k+2}, v_{l+1}, v_{l+2}\}\})$$

$$+P\left(T''', \bigcup_{\substack{(v_r,v_s)\in T''' \\ \{v_r,v_s\}\cap\{v_k,v_l\}=\varnothing}} \{\{v_i, v_j, v_r, v_s\}\} \cup \bigcup_{\substack{(v_r,v_s)\in T''' \\ \{v_r,v_s\}\cap\{v_{k-1},v_{l+1}\}=\varnothing}} \{\{v_{k-1}, v_{l+1}, v_r, v_s\}\}\right)$$

$$+P(T''', \{\{v_{k-1}, v_k, v_{l+1}, v_{l+2}\}, \{v_{k+2}, v_{k+1}, v_l, v_{l-1}\}, \{v_{k-1}, v_k, v_{k+2}, v_{k+1}\}, \{v_l, v_{l-1}, v_{l+1}, v_{l+2}\}\}).$$

Como podemos ver, a aresta (v_{k-1}, v_{l+1}) adicionada a T''' faz com que, ao calcular Δ_{p3}, os conjuntos de vértices seja diferentes aos verificados ao calcular Δ_{p2}. O complemento dos conjuntos usados para calcular Δ_{p2} e Δ_{p3} para encontrar as diferenças, nesse caso, contem todos os elementos da união:

$$\bigcup_{\substack{(v_r,v_s)\in T''' \\ \{v_r,v_s\}\cap\{v_{k-1},v_{l+1}\}=\varnothing}} \{\{v_{k-1}, v_{l+1}, v_r, v_s\}\}$$

Como número de conjuntos a ser calculado é $O(n)$, é impossível calcular Δ_{p3} em tempo constante caso $V_1 \cap V_2 \neq \varnothing$ e $V_1 \not\subset V_2$ dado as condições iniciais. □

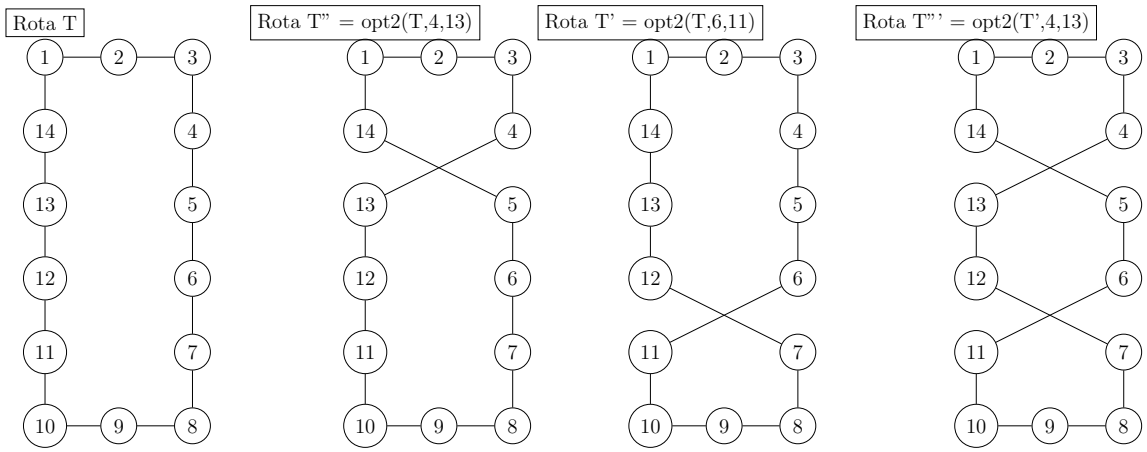

Figura 5.1: Movimentos 2-OPT para rota T=(1,..,14)

Exemplificando, considere a rota genérica T mostrada na figura 5.1. Seja T a solução corrente de uma iteração da abordagem *DILS* e T' e T'' vizinhos aprimorantes de T. Suponha que $\Delta_{p1} = DifProp(T, T')$, $\Delta_{p2} = DifProp(T, T'')$ sendo Δ_{p1} o melhor

valor para diferença entre as precondições violadas na vizinhança de T. Então, T' será selecionado para ser a solução corrente na próxima iteração. Continuando a busca local, ao buscar um novo vizinho aprimorante na vizinhança de T', teremos o vizinho T''', gerado pela operação $opt2(T', 6, 11)$, substituindo as arestas arestas (6,7) e (11,12) por (6,11) e (7,12) em T'. Então, o algoritmo precisa calcular $\Delta_{p3} = DifProp(T', T''')$ para verificar a viabilidade do movimento, mas sabemos que a quantidade de precondições violadas na maioria dos pares de arestas de T' e T''' já foram calculadas na iteração anterior. Dessa forma, para entender quais pares de arestas devem ser considerados para calcular $DifProp(T', T''')$ de forma eficiente, dado que já calculamos $DifProp(T, T'')$, montamos a tabela 5.1.

Tabela 5.1: Arestas verificadas para o calculo das proposições rota

DifProp(T,T")		DifProp(T',T"')	
$P(T)$	$P(T'')$	$P(T')$	$P(T''')$
(4,5),(1,2)	(4,13),(1,2)	(4,5),(1,2)	(4,13),(1,2)
(4,5),(2,3)	(4,13),(2,3)	(4,5),(2,3)	(4,13),(2,3)
*(4,5),(6,7)	*(4,13),(12,11)	*(4,5),(6,11)	*(4,13),(12,7)
(4,5),(7,8)	(4,13),(11,10)	(4,5),(11,10)	(4,13),(7,8)
(4,5),(8,9)	(4,13),(10,9)	(4,5),(10,9)	(4,13),(8,9)
(4,5),(9,10)	(4,13),(9,8)	(4,5),(9,8)	(4,13),(9,10)
(4,5),(10,11)	(4,13),(8,7)	(4,5),(8,7)	(4,13),(10,11)
*(4,5),(11,12)	*(4,13),(7,6)	*(4,5),(7,12)	*(4,13),(11,6)
(4,5),(12,13)	(4,13),(6,5)	(4,5),(12,13)	(4,13),(6,5)
(4,5),(13,14)	(4,13),(5,14)	(4,5),(13,14)	(4,13),(5,14)
(4,5),(14,1)	(4,13),(14,1)	(4,5),(14,1)	(4,13),(14,1)
(13,14),(1,2)	(5-14),(1,2)	(13,14),(1,2)	(5-14),(1,2)
(13,14),(2,3)	(5-14),(2,3)	(13,14),(2,3)	(5-14),(2,3)
(13,14),(3,4)	(5-14),(3,4)	(13,14),(3,4)	(5-14),(3,4)
(13,14),(5,6)	(5-14),(13,12)	(13,14),(5,6)	(5-14),(13,12)
*(13,14),(6,7)	*(5-14),(12,11)	*(13,14),(6,11)	*(5-14),(12,7)
(13,14),(7,8)	(5-14),(11,10)	(13,14),(11,10)	(5-14),(7,8)
(13,14),(8,9)	(5-14),(10,9)	(13,14),(10,9)	(5-14),(8,9)
(13,14),(9,10)	(5-14),(9,8)	(13,14),(9,8)	(5-14),(9,10)
(13,14),(10,11)	(5-14),(8,7)	(13,14),(8,7)	(5-14),(10,11)
*(13,14),(11,12)	*(5-14),(7,6)	*(13,14),(7,12)	*(5-14),(11,6)

Na primeira e na segunda coluna da tabela 5.1 mostramos os pares de arestas que são diferentes em T e T'' devido à operação $opt2(T, 4, 13)$. Portanto, para se calcular $DifProp(T, T'')$ deve-se calcular as precondições violadas nos pares de arestas da segunda coluna (T'') e subtrair precondições violadas nos pares de arestas da primeira coluna (T). Na terceira e na quarta coluna mostramos os pares de arestas que são diferentes em T' e T''' devido à operação $opt2(T', 4, 13)$. Então, para se calcular $DifProp(T', T''')$

deve- se calcular as precondições violadas nos pares de arestas da quarta coluna (T''') e subtrair precondições violadas nos pares de arestas da terceira coluna (T'). Observe que para maioria dos pares de arestas da terceira coluna as precondições violadas já foram calculadas ao verificarmos os pares de arestas da primeira coluna. Porém, na primeira coluna temos os pares de arestas: ((4,5),(6,7)), ((4,5),(11,12)), ((13,14),(6,7)) e ((13,14),(11,12)) que não estão presentes na terceira coluna (marcados com *) . Já na terceira coluna, tempos os pares de arestas: ((4,5),(6,11)), ((4,5),(7,12)), ((13,14),(6,11)) e ((13,14),(7,12)) que não estão presentes na primeira coluna (marcados com *). O mesmo pode ser observado entre a segunda e a quarta coluna nos pares de arestas marcados com *. Então, sabendo o valor de $\Delta_{p2} = DifProp(T, T'')$ podemos calcular $\Delta_{p3} = DifProp(T', T''')$ subtraindo de Δ_{p2} as precondições violadas pelas arestas da primeira coluna que não estão presentes na terceira coluna e somar o número de precondições violadas das arestas que estão presentes na terceira coluna mas não estão presente na primeira coluna, depois somar as precondições violadas pelas arestas da segunda coluna que não estão presentes na quarta coluna e somar o número de precondições violadas das arestas que estão presentes na quarta coluna mas não estão presente na terceira coluna. Observe que o calculo apresentado acima é o mesmo que calcular:

$$\Delta_{p3} = \Delta_{p2} - (P(T, v_k, v_{k+1}, v_i, v_{i+1}) + P(T, v_k, v_{k+1}, v_j, v_{j+1}) + P(T, v_l, v_{l+1}, v_i, v_{i+1}) + P(T, \{v_l, v_{l+1}, v_j, v_{j+1}\})) + (P(T', \{v_k, v_{k+1}, v_i, v_j\}) + P(T', \{v_k, v_{k+1}, v_{i+1}, v_{j+1}\}) + P(T', \{v_l, v_{l+1}, v_i, v_j\}) + P(T', \{v_l, v_{l+1}, v_{i+1}, v_{j+1}\})) + (P(T'', \{v_k, v_l, v_j, v_{j+1}\}) + P(T'', \{v_k, v_l, v_i, v_{i+1}\}) + P(T'', \{v_{k+1}, v_{l+1}, v_j, v_{j+1}\}) + P(T'', \{v_{k+1}, v_{l+1}, v_i, v_{i+1}\})) - (P(T'', \{v_k, v_l, v_{i+1}, v_{j+1}\}) + P(T'', \{v_k, v_l, v_i, v_j\}) + P(T'', \{v_{k+1}, v_{l+1}, v_{i+1}, v_{j+1}\}) + P(T'', \{v_{k+1}, v_{l+1}, v_i, v_j\})).$$

Como demonstrado no Caso 1 do Teorema 1.

Como visto no Teorema 1, dado $T = (v_1, v_2, ..., v_n, v_1)$ uma rota em $G(V, E)$, $T' = opt2(T, i, j)$, $T'' = opt2(T, k, l)$, $T''' = opt2(T', k, l)$, $\Delta_{p1} = DifProp(T, T')$, $\Delta_{p2} = DifProp(T, T'')$ não é possível calcular $\Delta_{p2} = DifProp(T', T''')$ em tempo constante se, $V_1 \cap V_2 \neq \varnothing$ e $V_1 \not\subset V_2$ onde $V_1 = \{v_i, ..., v_j\} \subseteq V$ e $V_2 = \{v_k, ..., v_l\} \subseteq V$. Como pode ser visto na figura 5.2, se consideramos $T''' = opt2(T', k+1, l)$ teremos as mesmas arestas removidas do movimento $opt2(T, k, l)$. Porém, as arestas adicionadas em $opt2(T', k+1, l)$ não poderiam ser adicionadas em $opt2(T, k, l)$ pois formariam dois ciclos disjuntos. Ou seja, se adicionássemos as mesmas arestas adicionadas em $opt2(T, k, l)$ na rota T''' teríamos uma solução inviável. Por outro lado, se, ao calcularmos $DifProp(T, T'')$, levássemos em conta a incidência das arestas paralelas (que desconectam a rota T), apesar da solução ser inviável, seria possível calcular $DifProp(T', T''')$ em tempo constante na próxima iteração.

Portanto, seja $opt2Par(T, i, j)$ a operação de remoção das arestas (v_i, v_{i+1}) e (v_j, v_{j+1}) e adição das arestas (v_i, v_{j+1}) e (v_{i+1}, v_j) na rota T gerando um grafo G' composto por dois ciclos disjuntos.

Teorema 2. *Dado* $T = (v_1, v_2, ..., v_n, v_1)$ *uma rota em* $G(V, E)$, $T' = opt2(T, i, j)$, $G' = opt2Par(T, k, l)$, $T''' = opt2(T', k, l)$, $\Delta_{p1} = DifProp(T, T')$, $\Delta_{p2} = DifProp(T, G')$ *é possível calcular* $\Delta_{p3} = DifProp(T', T''')$ *em tempo constante, se somente se,* $V_1 \cap V_2 \neq \varnothing$ *e* $V_1 \not\subset V_2$ *onde* $V_1 = \{v_i, ..., v_j\} \subseteq V$ *e* $V_2 = \{v_k, ..., v_l\} \subseteq V$ *e* $i > j$ *e* $k > l$.

A prova do Teorema 2 é análoga à prova do Teorema 1.

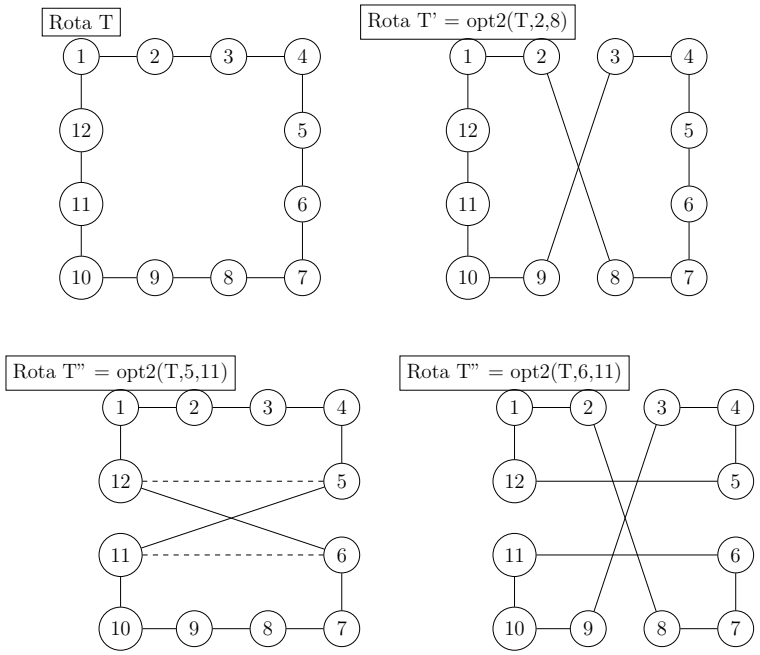

Figura 5.2: Movimentos 2-OPT para rota T=(1,..,12)

Com o que foi demostrado nos teoremas 1 e 2 aprimoramos o algoritmo DILS. Na nova versão, dado uma solução inicial T, a primeira iteração calcula todos os Δ_{pi} de todas as soluções aprimorantes s_i da vizinhança de T. Também é calculado todos Δ_{pj} das soluções inviáveis s_j geradas pelas operações $opt2Par(T, k, l)$ onde o custo dos pares de arestas removidos é maior que dos pares de arestas adicionados em T. Dessa forma, nas iterações seguintes são calculados os Δ_{pi} para a vizinhança da solução corrente em tempo constante para os casos cobertos nos teoremas 1 e 2. Ficando apenas os casos onde a operação 2-OPT inclui pelo menos uma das arestas adicionadas na última iteração sendo calculados em tempo $O(n)$, pois nesses casos não há Δ_{pi} calculado na iteração anterior que inclua as arestas permutadas.

5.3 Resultados computacionais

Com o intuito de avaliar a performance da abordagem *DILS* aprimorada, realizamos experimentos executando a nova versão do *DILS* aprimorado no mesmo conjunto de instâncias da versão anterior, partindo das mesmas soluções iniciais usadas nas execuções anteriores. Nesses testes obtivemos os mesmos resultados, porém, com um tempo de execução significativamente menor. Na tabela 5.2 mostramos os resultados para as mesmas instâncias da tabela 4.8. Para cada instância mostramos o custo da solução obtida pela abordagem *DILS* , o tempo de execução e o número de iterações. Comparando com os resultados obtidos com os da tabela 4.8 obtivemos os mesmos resultados com o tempo de execução significativamente menor. Porém, o tempo de execução em relação as abordagens *BILS* e *FILS* ainda é significativamente maior. Para a instância KroA200, por exemplo, o tempo de execução da abordagem *DILS* é aproximadamente 2000 vezes maior que o tempo de execução das abordagens *BILS* e *FILS* mostrados na tabela 4.8.

Tabela 5.2: Detalhamento dos resultados da busca DILS aprimorada. Executando 1 vez para cada instância a partir das mesmas soluções iniciais da tabela 4.8

Instância	DILS		
	Custo	Tempo	Iter.
kroA200	30393	95,56	7685
kroB200	31253	80,51	8388
pr152	79813	30,84	6303
rat195	2396	44,38	4999
tsp225	4133	76,72	7004

Diante dos resultados obtidos e apresentados na tabela 5.2 comparados com os mostrados na tabela 4.8, passamos a verificar o comportamento das abordagens *DILS*, *BILS* e *FILS*, fornecendo o mesmo tempo de execução para cada abordagem. Verificando, assim, se os resultados melhores obtidos pela abordagem *DILS* compensam o tempo maior de execução. Na tabela 5.3 mostramos as abordagens sendo executadas 10 vezes com soluções iniciais aleatórias por 300 segundos cada. Já na tabela 5.4 mostramos as abordagens sendo executadas 10 vezes com soluções iniciais geradas pela heurística de inserção do vizinho mais próximo, mostrada no algoritmo 4.2, por 300 segundos cada. A máquina utilizada em todos os testes é a mesma dos experimentos do capitulo 4.

Pode-se observar na tabela 5.3, onde os melhores valores estão destacados em negrito, que a abordagem *DILS* tem um desempenho pior se comparada às outras abordagens. Tal resultado pode ser explicado pelo pequeno número de execuções em relação as outras abordagens no mesmo intervalo de tempo. Pode-se observar que para a instância att48 a abordagem *DILS* faz por volta de 580 vezes menos execuções que as abordagens

BILS e *FILS*, para a instância KroA100 a abordagem *DILS* faz por volta de 1000 vezes menos execuções que as abordagens *BILS* e *FILS* e quanto mais vértices a instância tiver maior será a distância entre a media de execuções da abordagem *DILS* e as abordagens *BILS* e *FILS*. Tal distância entre o número de execuções das abordagens *BILS* e *FILS* e a abordagem *DILS* dá uma chance maior para que as abordagens *BILS* e *FILS* encontrem ótimos locais melhores no mesmo período de tempo.

Já na tabela 5.4, observando os melhores valores destacados em negrito, a bordagem *DILS* obteve todos os ótimos globais das instâncias testadas, enquanto a abordagem *BILS* obteve 9 ótimos globais das 11 instâncias testadas e a abordagem *FILS* obteve 7 ótimos globais das 11 instâncias testadas. Apesar da abordagem *DILS* ter executado menos vezes que as outras abordagens no mesmo período de tempo obteve uma performance igual ou melhor que as outras abordagens. Por exemplo, para as instâncias KroA150 e KroB150 a abordagem *DILS* foi a única a encontrar os ótimos globais, com tal resultado obtido com, em média, 10 e 13 vezes menos execuções que as abordagens *BILS* e *FILS* respectivamente. Nota-se que o número de execuções da abordagem *DILS* para um mesmo período de tempo é significativamente maior quando iniciado com soluções geradas a partir de heurísticas construtivas se comparado a quando iniciado a partir de soluções aleatórias. Tal observação pode ser explicada pelo fato de que soluções aleatórias tendem a ser piores que soluções geradas por heurísticas e, portanto, precisam de mais passos até um ótimo local acarretando em maior tempo de execução. Ao inciar a partir de soluções melhores e a cada passo evitar cair em ótimos locais, porém, já estando próximo de soluções boas, a abordagem *DILS* obtém vantagem sobre as outras abordagens. No entanto, ao iniciar a partir de soluções aleatórias a abordagem *DILS* gasta tempo desnecessário evitando ótimos locais estando ainda longe deles, o que acarreta em muito mais passos até as soluções realmente boas e, consequentemente, demora muito tempo para convergir.

Tabela 5.3: Detalhamento dos resultados das buscas DILS, 2-OPT Melhor Ap. e 2-OPT Primeiro Ap. Executando por 300 segundos, 10 repeticoes,a partir de soluções iniciais aleatórias.

Instância	2-OPT Melhor Ap.					Ótimo	E. Média	E. Melhor
	T. Médio	Exe.	C. Médio	C. Mín.	C. Máx.			
att48	300,00	127651,60	**10628,00**	**10628**	**10628**	10.628,00	0,00%	0,00%
berlin52	300,00	101865,80	**7542,00**	**7542**	**7542**	7.542,00	0,00%	0,00%
kroA100	300,01	12256,00	**21287,40**	**21282**	**21296**	21.282,00	0,03%	0,00%
kroA150	300,04	3578,30	26859,30	**26596**	27037	26.524,00	1,26%	0,27%
kroB150	300,04	3567,80	26439,10	**26195**	26581	26.130,00	1,18%	0,25%
kroC100	300,01	12274,30	20754,70	**20749**	20798	20.749,00	0,03%	0,00%
kroD100	300,01	12521,60	**21356,50**	**21294**	**21450**	21.294,00	0,29%	0,00%
kroE100	300,01	12563,30	**22106,40**	**22068**	**22162**	22.068,00	0,17%	0,00%
lin105	300,01	10368,20	**14379,00**	**14379**	**14379**	14.379,00	0,00%	0,00%
pr107	300,02	9995,40	**44365,90**	**44303**	**44442**	44.303,00	0,14%	0,00%
pr124	300,03	5913,50	**59030,00**	**59030**	**59030**	59.030,00	0,00%	0,00%
						Erro Médio	0,59%	0,37%
	2-OPT Primeiro Ap.							
	T. Médio	Exe.	C. Média	C. Mín.	C. Máx.			
att48	300,00	130093,80	**10628,00**	**10628**	**10628**	10.628,00	0,00%	0,00%
berlin52	300,00	111704,20	**7542,00**	**7542**	**7542**	7.542,00	0,00%	0,00%
kroA100	300,01	10586,90	**21282,00**	**21282**	**21282**	21.282,00	0,00%	0,00%
kroA150	300,04	2746,70	**26759,10**	26653	**26836**	26.524,00	0,89%	0,49%
kroB150	300,06	2758,30	**26337,20**	26273	**26421**	26.130,00	0,79%	0,55%
kroC100	300,01	10472,90	**20749,00**	**20749**	**20749**	20.749,00	0,00%	0,00%
kroD100	300,01	10725,00	21408,60	21307	21477	21.294,00	0,54%	0,06%
kroE100	300,02	10856,50	22143,00	22073	22185	22.068,00	0,34%	0,02%
lin105	300,01	9067,70	14386,70	**14379**	14412	14.379,00	0,05%	0,00%
pr107	300,01	10172,40	44461,20	44347	44526	44.303,00	0,36%	0,10%
pr124	300,03	5660,10	59042,40	**59030**	59092	59.030,00	0,02%	0,00%
						Erro Médio	0,58%	0,43%
	DILS							
	T. Médio	Exe.	C. Média	C. Mín.	C. Máx.			
att48	300,54	221,60	10641,00	**10628**	10658	10.628,00	0,12%	0,00%
berlin52	301,06	154,90	7550,50	**7542**	7627	7.542,00	0,11%	0,00%
kroA100	312,90	11,70	21601,40	21381	21925	21.282,00	1,50%	0,47%
kroA150	373,20	3,00	27231,20	26737	27520	26.524,00	2,67%	0,80%
kroB150	368,21	3,00	27133,80	26667	27854	26.130,00	3,84%	2,06%
kroC100	314,03	12,10	21060,10	20901	21151	20.749,00	1,50%	0,73%
kroD100	318,01	12,00	21714,00	21466	21977	21.294,00	1,97%	0,81%
kroE100	314,08	12,30	22402,20	22111	22688	22.068,00	1,51%	0,19%
lin105	311,09	9,10	14535,80	14502	14605	14.379,00	1,09%	0,86%
pr107	316,33	9,00	46177,80	45326	46673	44.303,00	4,23%	2,31%
pr124	327,01	5,00	60184,70	59087	61133	59.030,00	1,96%	0,10%
						Erro Médio	2,04%	1,02%

Na Tabela 5.5 relatamos o custo médio das soluções encontradas nas dez execuções (Avg. Z) e o custo da melhor solução nas dez execuções (Min. Z). Quando consideramos o

Tabela 5.4: Detalhamento dos resultados das buscas DILS, 2-OPT Melhor Ap. e 2-OPT Primeiro Ap. Executando por 300 segundos, 10 repetições, para cada instância a partir de soluções geradas por heurística construtiva.

Instância	2-OPT Melhor Ap.					Ótimo	E. Média	E. Melhor
	T. Médio	Exe.	C. Médio	C. Mín.	C. Máx.			
att48	300,00	646141,70	**10628,00**	**10628**	**10628**	10.628,00	0,00%	0,00%
berlin52	300,00	576040,90	**7542,00**	**7542**	**7542**	7.542,00	0,00%	0,00%
kroA100	300,00	93306,40	21284,10	**21282**	21296	21.282,00	0,01%	0,00%
kroA150	300,00	33037,60	**26695,30**	26596	27037	26.524,00	0,65%	0,27%
kroB150	300,01	31845,40	**26215,90**	26165	26581	26.130,00	0,33%	0,13%
kroC100	300,00	93336,80	20753,30	**20749**	20798	20.749,00	0,02%	0,00%
kroD100	300,00	99233,90	**21314,60**	**21294**	**21344**	21.294,00	0,10%	0,00%
kroE100	300,00	99064,70	**22090,70**	**22068**	**22111**	22.068,00	0,10%	0,00%
lin105	300,00	127074,40	**14379,00**	**14379**	**14379**	14.379,00	0,00%	0,00%
pr107	300,00	160115,20	**44303,00**	**44303**	**44303**	44.303,00	0,00%	0,00%
pr124	300,00	111683,00	**59030,00**	**59030**	**59030**	59.030,00	0,00%	0,00%
						Erro Médio	0,11%	0,04%
	2-OPT Primeiro Ap.							
	T. Médio	Exe.	C. Média	C. Mín.	C. Máx.			
att48	300,00	1034250,20	**10628,00**	**10628**	**10628**	10.628,00	0,00%	0,00%
berlin52	300,00	839930,80	**7542,00**	**7542**	**7542**	7.542,00	0,00%	0,00%
kroA100	300,00	132670,70	**21282,00**	**21282**	**21282**	21.282,00	0,00%	0,00%
kroA150	300,00	44406,20	26695,70	26637	**26797**	26.524,00	0,65%	0,43%
kroB150	300,00	44073,60	26279,90	26231	26347	26.130,00	0,57%	0,39%
kroC100	300,00	131138,60	**20749,00**	**20749**	**20749**	20.749,00	0,00%	0,00%
kroD100	300,00	142148,30	21325,80	21307	21358	21.294,00	0,15%	0,06%
kroE100	300,00	139057,50	22094,00	22073	22120	22.068,00	0,12%	0,02%
lin105	300,00	180728,80	**14379,00**	**14379**	**14379**	14.379,00	0,00%	0,00%
pr107	300,00	227934,60	**44303,00**	**44303**	**44303**	44.303,00	0,00%	0,00%
pr124	300,00	147996,00	**59030,00**	**59030**	**59030**	59.030,00	0,00%	0,00%
						Erro Médio	0,14%	0,08%
	DILS							
	T. Médio	Exe.	C. Média	C. Mín.	C. Máx.			
att48	300,00	52471,60	**10628,00**	**10628**	**10628**	10.628,00	0,00%	0,00%
berlin52	300,01	50713,80	**7542,00**	**7542**	**7542**	7.542,00	0,00%	0,00%
kroA100	300,02	9267,20	**21282,00**	**21282**	**21282**	21.282,00	0,00%	0,00%
kroA150	300,07	3241,50	26735,10	**26524**	26825	26.524,00	0,80%	0,00%
kroB150	300,06	3282,90	26259,90	**26130**	**26344**	26.130,00	0,50%	0,00%
kroC100	300,01	9005,30	**20749,00**	**20749**	**20749**	20.749,00	0,00%	0,00%
kroD100	300,01	9360,60	21386,50	**21294**	21423	21.294,00	0,43%	0,00%
kroE100	300,02	8862,30	22092,40	**22068**	22121	22.068,00	0,11%	0,00%
lin105	300,01	16034,40	14383,40	**14379**	14401	14.379,00	0,03%	0,00%
pr107	300,01	27381,30	**44303,00**	**44303**	**44303**	44.303,00	0,00%	0,00%
pr124	300,01	18547,00	**59030,00**	**59030**	**59030**	59.030,00	0,00%	0,00%
						Erro Médio	0,17%	0,00%

custo mínimo ao longo das dez execuções para cada instância, o DILS obteve os melhores resultados para 25 instâncias, enquanto o BILS e o FILS encontraram as melhores soluções

para apenas 10 instâncias cada.

Fizemos, então, testes de avaliação de desempenho considerando os gráficos *time to target* (TTT) [2]. Usamos um limite de tempo geral de 900 segundos e executamos 100 vezes, finalizando ao atingir a meta escolhida ou atingindo o tempo limite da execução. As figuras 5.3 a 5.5 mostram gráficos TTT para as instâncias rat195, tsp225 e rd400. Quatro gráficos são mostrados para cada instância, considerando alvos progressivamente mais difíceis. O alvo mais difícil é definido para o melhor custo médio da solução sobre os três algoritmos na Tabela 5.5 arredondado para o inteiro mais próximo. Os outros três alvos estão 1%, 2% e 3% acima do alvo mais difícil, arredondado para o número inteiro mais próximo.

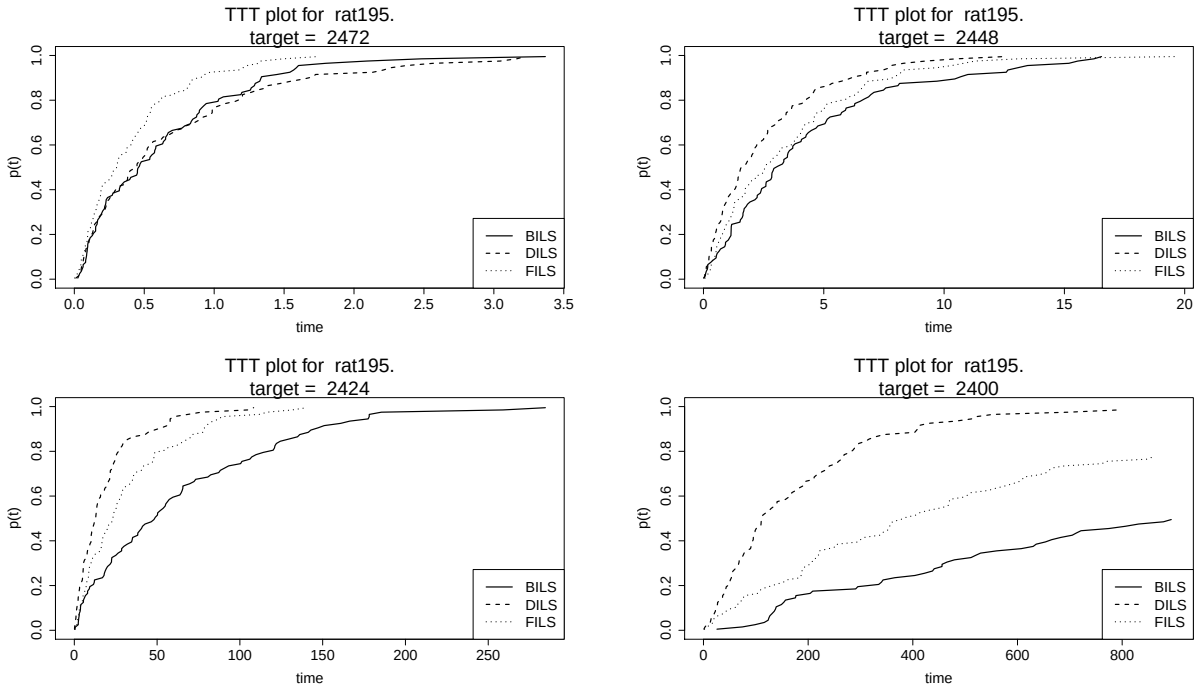

Figura 5.3: TTT plots for rat195, with easiest target on the top left and the hardest target on the bottom right

Os gráficos mostram que quando o alvo é fácil BILS e FILS têm melhor desempenho do que DILS. No entanto, todos os três algoritmos alcançam o alvo em alguns segundos. Por outro lado, quando a meta fica mais difícil de atingir o desempenho de DILS melhora em relação a BILS e FILS. Com o alvo mais difícil, o desempenho superior do DILS é evidente. Podemos concluir que, enquanto o DILS é mais lento do que as estratégias clássicas para obter soluções de qualidade média, no longo prazo tende a alcançar soluções melhores do que as outras abordagens.

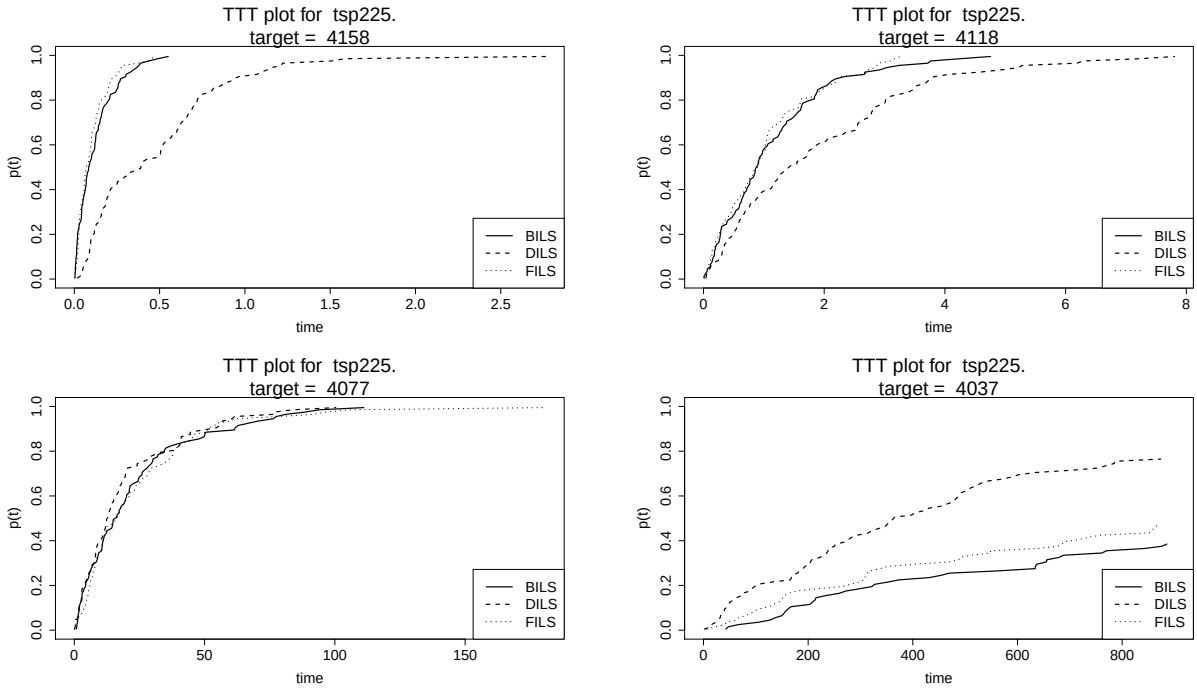

Figura 5.4: TTT plots for tsp225, with easiest target on the top left and the hardest target on the bottom right

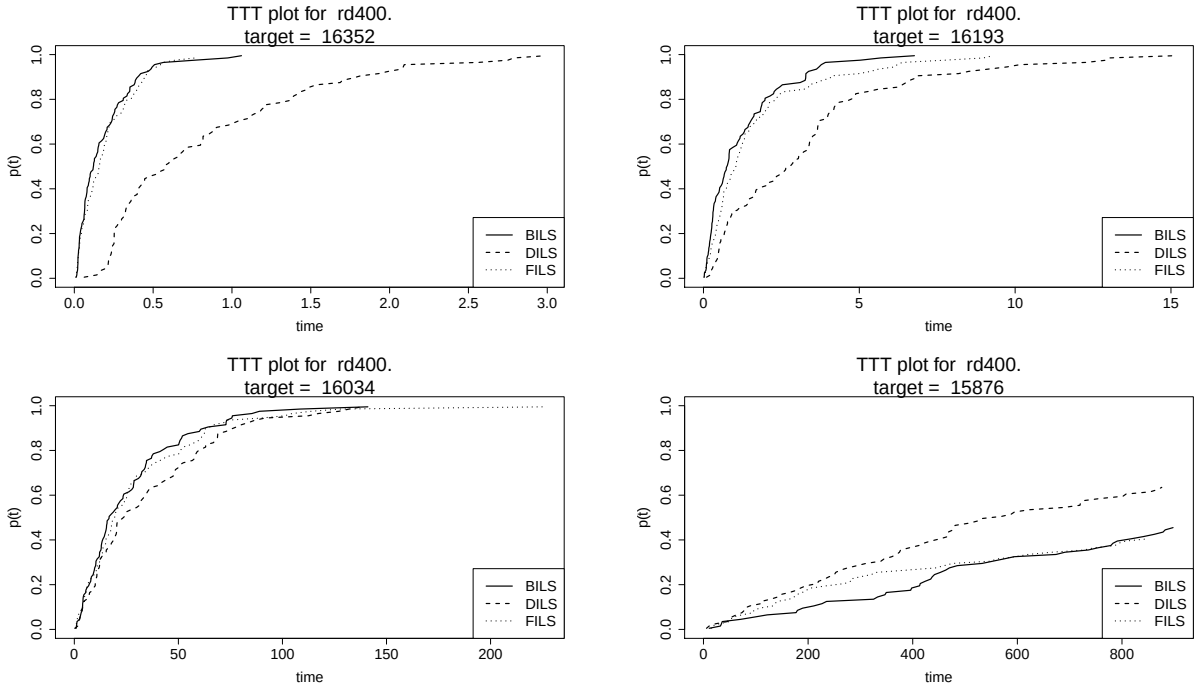

Figura 5.5: TTT plots for rd400, with easiest target on the top left and the hardest target on the bottom right

Tabela 5.5: Resultados para DILS, BILS e FILS. Executando por 300 segundos, 10 repetições, partindo da solução gerada a partir da heurística construtiva. A melhor solução obtida para cada instância é destacada em **Negrito**

	Instância	BILS		FILS		DILS	
		Avg. Z	Min. Z	Avg. Z	Min. Z	Avg. Z	Min. Z
1	berlin52	**7542.0**	**7542**	**7542.0**	**7542**	**7542.0**	**7542**
2	ch130	6117.6	**6110**	**6117.0**	**6110**	6120.4	**6110**
3	ch150	6577.3	6561	**6566.0**	**6548**	6574.9	**6548**
4	d1291	**55200.8**	54935	55401.4	54961	55287.6	**54901**
5	kroA200	**29678.0**	**29573**	29686.2	29599	29683.3	29610
6	kroB200	**29728.2**	29645	29733.3	**29632**	29793.0	**29632**
7	kroC100	**20749.0**	**20749**	**20749.0**	**20749**	**20749.0**	**20749**
8	kroD100	**21294.0**	**21294**	21296.8	**21294**	21308.4	**21294**
9	lin105	**14379.0**	**14379**	**14379.0**	**14379**	**14379.0**	**14379**
10	linhp318	**43245.2**	43103	43255.2	43059	43245.9	**42958**
11	p654	**35049.4**	35007	35069.7	35049	35069.4	**34915**
12	pcb1173	62440.5	61885	62175.9	61866	**61765.1**	**61141**
13	pcb442	53506.1	53383	53495.1	53117	**53038.3**	**52715**
14	pr1002	275906.2	274720	276056.4	274975	**275679.4**	**274353**
15	pr152	**73682.0**	**73682**	**73682.0**	**73682**	**73682.0**	**73682**
16	pr2392	416929.7	414049	**415425.0**	**412988**	415618.1	413393
17	pr299	49185.2	**48943**	**49179.5**	49011	49220.8	49117
18	pr439	110308.6	109730	110489.4	109937	**110048.8**	**109394**
19	rat195	2405.2	2387	2402.2	**2380**	**2399.8**	2389
20	rd400	15908.6	15856	15904.4	15835	**15875.7**	**15811**
21	rl1323	290386.3	**285853**	291240.3	287928	**289982.0**	**285853**
22	rl1889	346631.9	343891	348353.9	346499	**344231.4**	**339537**
23	tsp225	**4036.9**	4031	4041.2	4022	4037.5	**4020**
24	u1060	238396.8	237519	238504.0	237602	**237519.0**	**236893**
25	u1432	**166572.3**	**165792**	166646.2	166054	166621.3	166282
26	u1817	64391.0	64083	64514.0	64137	**64380.7**	**63945**
27	u2152	73100.0	72417	73213.5	72675	**72780.1**	**72361**
28	u2319	248396.3	247856	**248218.4**	247735	248246.1	**247533**
29	u574	38865.3	38745	38821.6	38597	**38807.8**	**38548**
30	vm1748	363912.0	361293	363577.3	362488	**362627.3**	**358457**

Capítulo 6

DILS aplicado ao problema do corte máximo

Neste capítulo, descreveremos o uso da abordagem *DILS* na busca de soluções boas para o problema MAX-CUT e apresentaremos os resultados obtidos.

O problema do Corte Máximo (MAX-CUT) pode ser definido como o seguinte: dado um gráfico $G = (V, E)$, divida V em conjuntos S e $\bar{S}$ de modo que o número de arestas com uma extremidade em S e o outra em $\bar{S}$ é maximizado. O problema é NP-Difícil [22]. Heurísticas, algoritmos de aproximação e abordagens exatas foram propostas para o problema do corte máximo. A maior parte da literatura considera a versão ponderada do problema, em que a soma dos pesos das arestas com os vértices em ambos os conjuntos deve ser maximizada, enquanto consideramos o problema não ponderado. Uma das heurísticas proposta para o problema é a busca local usando a vizinhança de 1-Flip proposta por [20].

A vizinhança de 1 flip consiste em todas as soluções que podem ser obtidas de uma solução s movendo um vértice de S para $\bar{S}$ ou de $\bar{S}$ para S. A diferença entre o custo de uma solução s e uma solução vizinha s', obtida movendo um vértice v de um conjunto para outro, é dada pela diferença entre o número de vértices adjacentes a v que estão no mesmo conjunto de v e o número de vértices adjacentes a v que estão no outro conjunto.

A abordagem *DILS* aplicada ao 1-Flip considera a precondição de otimalidade sobre a vizinhança 1-Flip apresentada por [40]. Para selecionar as soluções viáveis que tenham menor possibilidade de conduzirem a ótimos locais, o algoritmo verifica se uma dada solução viola alguma das precondições apresentadas. Assim como ocorreu na busca 2-OPT, quanto maior for o número precondições violadas por uma solução candidata, espera-se que mais distante esteja de um ótimo local, ou seja, mais movimentos serão necessários para se tornar um ótimo local. Esse processo é apresentado no algoritmo 6.1.

Em um ótimo local para a vizinhança de 1-flip nenhum vértice pode ter mais vértices adjacentes em seu próprio conjunto do que no outro conjunto, caso contrário, ao mover o vértice para o outro conjunto melhoraria a solução.

Seja $\delta(v)$ os vértices adjacentes a v em G e $T(s, v)$ os vértices de G que estão no mesmo conjunto que v na solução s. Então, o verificador de precondição p verifica, para um dado $v \in V$, se $|\{u : u \in \delta(v), u \in T(s, v)\}| \leq |\delta(v)|/2$.

O algoritmo 6.1 pega uma solução inicial e itera até que nenhum vizinho aprimorante seja encontrado. Em cada iteração, para cada vizinho aprimorante, o algoritmo calcula o número de vezes que a precondição p é violada. O vizinho aprimorante com o maior número de precondições violadas é selecionado como a próxima solução a ser visitada. Empates são resolvidos pelo melhor valor para a função objetivo.

Algoritmo 6.1: Busca Local 1-FlipDILS.

```
1  Busca Local 1-FlipDILS(s)
2  enquanto I(s) = {s' ∈ N(s) : f(s') < f(s)} não for vazio faça
3      k ← −1;
4      c ← ∞;
5      para cada s' ∈ I(s) faça
6          k' ← 0;
7          para cada v ∈ V faça
8              k' ← k' + (p(v) = false);
9          se k' > k or (k' = k and f(s') < c) então
10             s'' ← s';
11             c ← f(s');
12             k ← k';
13     s ← s'';
14 return s;
```

6.1 Detalhes de Implementação

Uma implementação óbvia da estratégia do melhor aprimorante ou da estratégia do primeiro aprimorante usando a vizinhança 1-flip seria armazenar, para cada vértice, a diferença entre o número de vértices adjacentes no mesmo conjunto e o número de vértices adjacentes no outro conjunto. Essa diferença, que chamamos de $\Delta(v)$, é a melhoria do valor da função objetivo se v for movido para o outro conjunto.

Essa implementação pode ser melhorada computando e atualizando uma lista de vértices para a qual Δ é maior que 0. Todos os vizinhos aprimorantes podem ser obtidos movendo um vértice desta lista. Ao trabalhar com soluções relativamente boas, como aquelas produzidas por uma heurística de construção gulosa, o número de vértices com Δ positivo é muito menor do que o número de vértices do grafo. Então, na estrategia primeiro aprimorante, pode-se simplesmente escolher o primeiro vértice da lista, enquanto, na estratégia melhor aprimorante, pode-se procurar o vértice na lista com o valor máximo de Δ.

Pode-se pensar que a melhor maneira de implementar o melhor aprimorante é mantendo uma lista ordenada de vértices com $\Delta > 0$. No entanto, ao trabalhar com grafos de densidade em torno de 0,5, correspondendo às instâncias mais interessantes para a versão do problema do corte máximo considerado aqui, os valores de Δ mudam para cerca de metade dos vértices após cada movimento. Portanto, é mais barato fazer a varredura de uma lista inteira não classificada a cada iteração em vez de atualizar uma lista classificada após cada iteração.

Para implementar o estratégia melhor aprimorante, precisamos calcular, além do Δ, o número de vértices que teriam $\Delta > 0$ se o vértice v fosse movido. Observe que esse número é exatamente o número de vezes que a precondição é violada. Em vez de calcular esse número diretamente calculamos a diferença em precondições violadas de cada solução vizinha aprimorante em relação à solução atual, à qual nos referimos como Δ'.

Como é mais difícil atualizar Δ' do que Δ. Após cada movimento, Δ só é atualizado para o vértice que está sendo movido e seus vértices adjacentes. No entanto, Δ' pode mudar também para os vértices adjacentes aos vértices adjacentes ao vértice movido. Em grafos de alta densidade as operações de atualização envolveriam quase todos os vértices do grafo. No entanto, Δ' só é relevante para melhorar os vizinhos, ou seja, os vizinhos obtidos movendo um vértice com um Δ positivo. Então, em nossa proposta calculamos Δ' apenas para os vértices com Δ positivo.

Para calcular Δ' usamos uma lista adicional de vértices que têm $\Delta \in \{-1, 0, 1, 2\}$. Esses vértices são aqueles que podem ir de um Δ positivo para um Δ não positivo ou de um Δ não positivo para um Δ positivo após o movimento de um vértice adjacente e, portanto, precisam ser considerados no cálculo do valor Δ' de seus vértices adjacentes.

6.2 Configuração dos testes

Para investigar o comportamento de DILS aplicado a vizinhaça 1-FLIP executamos DILS, BILS e FILS em uma única instância de teste, partindo da mesma solução inicial. Nesses testes, calculamos o número de precondições violadas em cada iteração para todos os três métodos. Isso nos permite mostrar como os três métodos diferem em termos de suas progressões de qualidade da solução e o número de precondições violadas em cada iteração da pesquisa. Além disso, para investigar o comportamento, analisamos o número de iterações realizadas para atingir um ótimo local por cada um dos métodos.

Ao analisar o desempenho do DILS consideramos os métodos de busca local em uma estrutura *multi-start*. Cada método é reiniciado ao atingir um ótimo local enquanto registra o valor do melhor ótimo local encontrado. O consumo de tempo de DILS pode ser

maior do que o consumo de tempo para os outros métodos devido ao fato de ter um maior esforço computacional por iteração ao tentar executar mais iterações antes de convergir para um ótimo local. Portanto, ao avaliar o desempenho do DILS, é injusto executar cada um dos métodos para qualquer número fixo de reinicializações.

Em vez disso, consideramos duas configurações diferentes para fazer comparações justas. Na primeira configuração fixamos um limite de tempo de execução e executamos repetidamente uma determinada pesquisa local (BILS, FILS ou DILS) a partir de diferentes soluções iniciais até que o limite de tempo seja atingido. A melhor solução encontrada é retornada. Esta abordagem *multi-start* é uma comparação mais justa dos métodos, uma vez que os métodos mais rápidos podem ser repetidos mais vezes do que os mais lentos, aumentando, assim, a probabilidade de encontrar um bom ótimo local.

Na segunda configuração consideramos instâncias individuais e produzimos gráficos *time to target* (TTT) [2]. As instâncias para os teste foram geradas aleatoriamente e consistem em grafos com o número de nós (|V |) variando de 1500 a 2500 e uma densidade (D) variando de 0,45 a 0,54.

Nos teste do para o problema max-cut usamos uma heurística de construção que permuta aleatoriamente os nós do grafo e, na ordem obtida, adiciona o próximo nó ao melhor dos dois conjuntos considerando apenas os nós previamente incluídos na solução.

Os métodos de busca local foram implementados usando a linguagem de programação C. O código foi compilado com o compilador gcc usando o sinalizador de otimização O3. Os experimentos foram executados em um Intel (R) Xeon (R) E5405 de 64 bits, máquina de 2 GHz com 4 processadores e 16,0 GB de RAM executando o sistema operacional Ubuntu 18.04.3 LTS. Os métodos foram todos programados de acordo com os mesmos padrões de programação e com o mesmo nível de otimização.

6.3 Resultados computacionais

Para investigar o comportamento da abordagem DILS realizamos o mesmo tipo de testes do TSP. Executamos DILS, BILS e FILS para a mesma instância e a partir da mesma solução inicial. A Figura 6.1 mostra o número de precondições violadas pela solução atual a cada iteração em uma única execução de cada busca em uma instância com 2.000 vértices e densidade de 0,5, enquanto a Figura 6.2 considera uma instância com 2500 vértices e a mesma densidade. As soluções iniciais foram obtidas com heurística de construção. As figuras 6.3 e 6.4 mostram, para as mesmas execuções de cada pesquisa, o valor da função objetivo a cada iteração da busca.

Como ocorreu no TSP, o número de iterações do DILS é maior do que para os

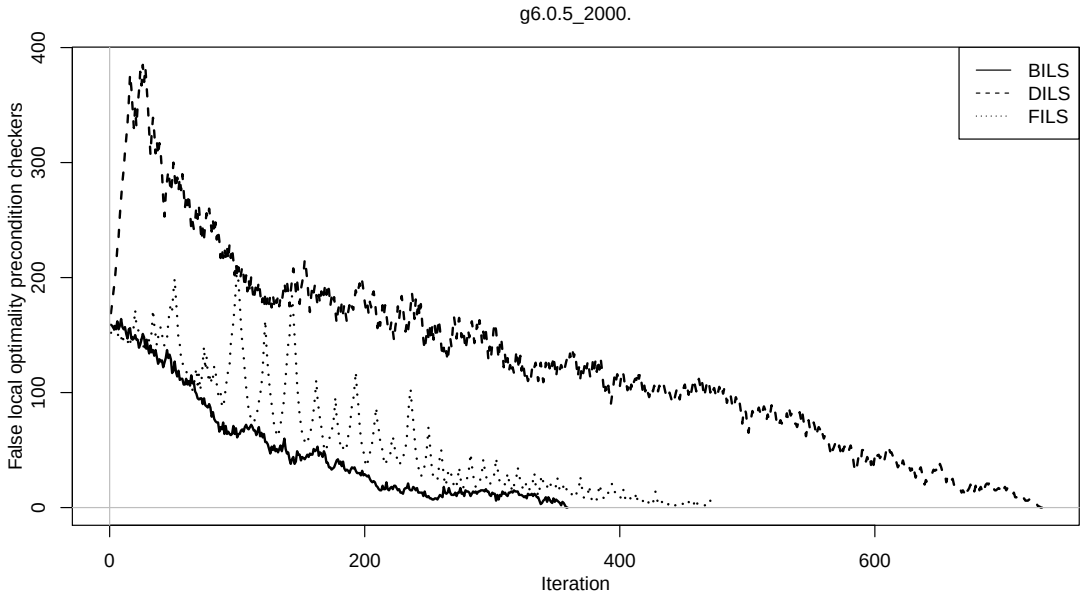

Figura 6.1: Número de precondições não satisfeitas pela solução atual para uma instância com 2.000 vértices e densidade 0,5, com a solução inicial criada por uma heurística construtiva

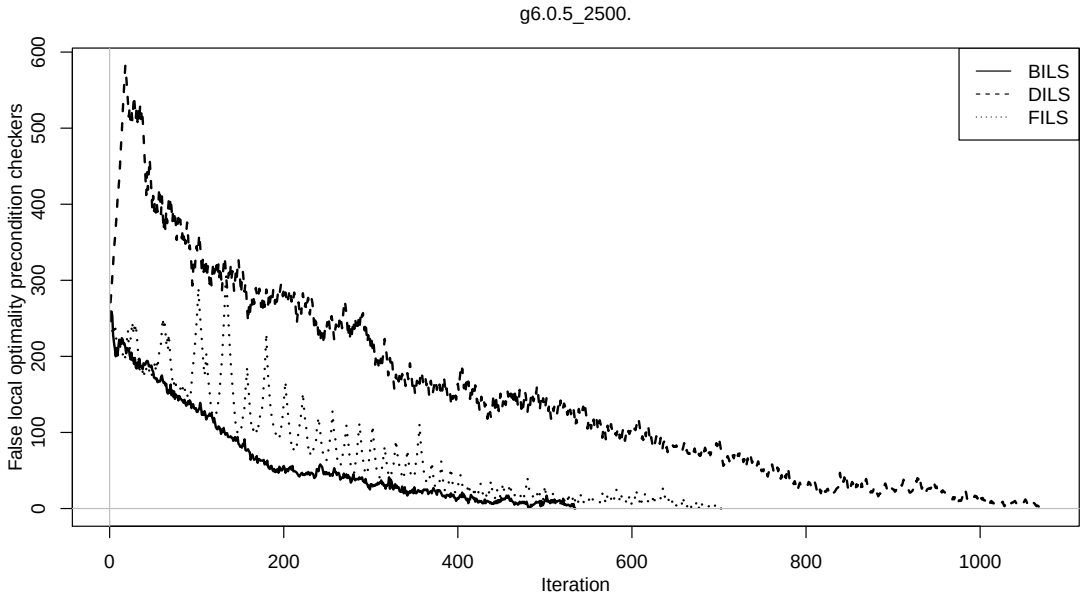

Figura 6.2: Número de precondições não satisfeitas pela solução atual para uma instância com 2500 vértices e densidade 0,5, com a solução inicial criada por uma heurística construtiva

outros métodos. Nas primeiras iterações da pesquisa o DILS passa para soluções com aproximadamente três vezes o número de precondições violadas em comparação com a

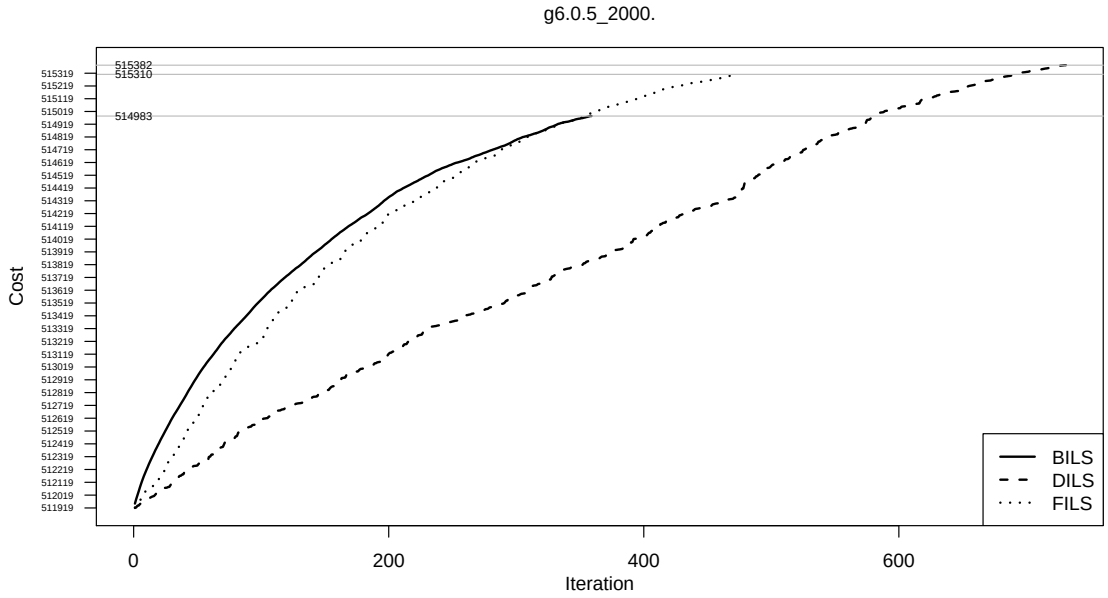

Figura 6.3: Valor da função objetivo da solução atual para uma instância com 2.000 vértices e densidade 0,5, com a solução inicial criada por uma heurística construtiva

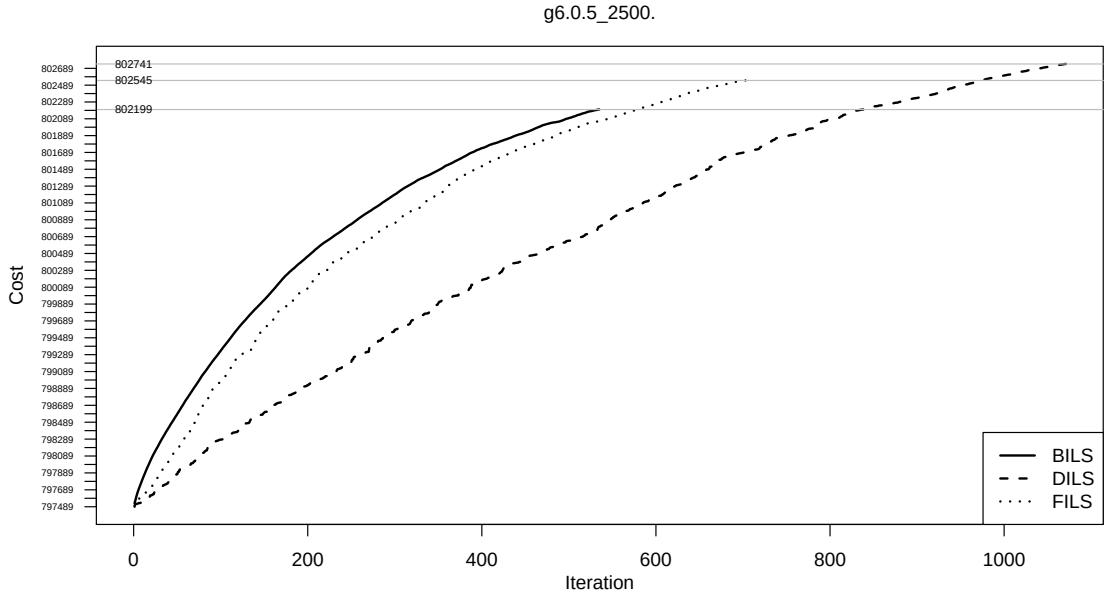

Figura 6.4: Valor da função objetivo da solução atual para uma instância com 2500 vértices e densidade 0,5, com a solução inicial criada por uma heurística construtiva

solução inicial. Então, DILS converge lentamente e encontra melhores ótimos locais do que BILS e FILS.

Tabela 6.1: Número de iterações realizadas por cada busca. Média de 100 execuções.

	|V|	D	BILS	FILS	DILS	DILS/BILS
1	1500	0.45	364.10	402.00	628.40	1.73
2	1500	0.46	346.30	416.20	624.50	1.80
3	1500	0.47	349.40	400.50	620.80	1.78
4	1500	0.48	331.20	417.90	627.60	1.89
5	1500	0.49	320.40	379.70	596.30	1.86
6	1500	0.50	345.90	385.50	609.10	1.76
7	1500	0.51	344.90	410.90	601.30	1.74
8	1500	0.52	332.50	404.80	611.30	1.84
9	1500	0.53	356.40	413.20	609.70	1.71
10	1500	0.54	324.60	422.80	617.10	1.90
11	2000	0.45	517.00	530.30	887.40	1.72
12	2000	0.46	495.30	575.90	870.60	1.76
13	2000	0.47	491.20	585.00	894.10	1.82
14	2000	0.48	431.70	596.00	901.00	2.09
15	2000	0.49	430.80	554.90	835.90	1.94
16	2000	0.50	450.60	615.70	865.60	1.92
17	2000	0.51	505.50	578.30	876.60	1.73
18	2000	0.52	463.90	533.00	863.30	1.86
19	2000	0.53	469.70	574.40	876.40	1.87
20	2000	0.54	472.20	554.90	885.20	1.87
21	2500	0.45	658.80	747.10	1132.80	1.72
22	2500	0.46	654.00	757.00	1149.80	1.76
23	2500	0.47	644.20	751.20	1161.70	1.80
24	2500	0.48	572.90	696.70	1160.30	2.03
25	2500	0.49	672.90	798.10	1162.80	1.73
26	2500	0.50	611.80	757.20	1157.40	1.89
27	2500	0.51	633.90	698.30	1104.70	1.74
28	2500	0.52	626.10	751.10	1138.00	1.82
29	2500	0.53	634.30	725.70	1091.50	1.72
30	2500	0.54	569.90	727.80	1073.20	1.88

A tabela 6.1 mostra o número médio de iterações para cada método em 100 execuções. O número de iterações do DILS é novamente próximo ao dobro do número de iterações do BILS em todas as instâncias consideradas, com uma proporção média de 1,82.

6.4 Performance da Busca

Assim como no TSP, mostramos resultados para corte máximo e vizinhança de 1 flip usando um limite de tempo fixo e um alvo fixo. A tabela 6.2 mostra os resultados da execução de DILS, BILS e FILS por 60 segundos, executando cada método *multi-start* dez vezes, partindo de soluções geradas pela heurística de construção descrita.

O desempenho superior do DILS é evidente. Para 26 das 30 instâncias DILS obteve a melhor solução, enquanto FILS obteve a melhor solução em três instâncias e BILS obteve a melhor solução apenas para uma instância. Aqui, o comportamento médio é ainda mais favorável ao DILS, pois o método tem o melhor desempenho médio em todas as 30 instâncias, enquanto os outros dois métodos falham em obter o melhor desempenho médio em qualquer instância.

Nos testes de avaliação de desempenho, produzimos gráficos *time to target* (TTT). As figuras 6.5, 6.6 e 6.7 mostram gráficos TTT para instâncias com 1500, 2000 e 2500 nós. A densidade de cada gráfico é 0,5. O limite de tempo geral foi definido para 180 segundos. Três gráficos são mostrados para cada instância, considerando alvos progressivamente mais difíceis. O alvo mais difícil é definido para o melhor valor médio da função objetivo, considerando cada método separadamente, na Tabela 6.2 menos 0,05 % arredondado para o inteiro mais próximo. As outras duas metas estão 0,1 % e 0,15 % abaixo da melhor média, arredondado para o número inteiro mais próximo.

Os gráficos mostram que, com alvos fáceis e grafos grandes, BILS e FILS superam DILS. No entanto, nesses casos, todos os três algoritmos alcançam o alvo em poucos segundos. Por outro lado, quando a meta fica mais difícil de atingir, o desempenho do DILS é muito melhor do que o do BILS e do FILS.

Tabela 6.2: Resultados da execução de DILS, BILS e FILS por 60 segundos, 10 repetições, a partir de soluções geradas pela heurística construtiva. A melhor solução obtida para cada instância é destacada em **Negrito**.

	|V|	D	BILS		FILS		DILS	
			Avg. Z	Max. Z	Avg. Z	Max. Z	Avg. Z	Max. Z
1	1500	0.45	263644.1	263785	263717.2	263864	**263875.1**	**263958**
2	1500	0.46	268781.1	268869	268795.2	268982	**268911.9**	**269020**
3	1500	0.47	274362.1	274433	274381.1	274422	**274547.7**	**274714**
4	1500	0.48	280477.1	280542	280549.7	280631	**280675.1**	**280770**
5	1500	0.49	285535.6	285671	285574.3	285641	**285690.0**	**285852**
6	1500	0.50	290937.2	291142	291005.4	291163	**291123.5**	**291256**
7	1500	0.51	297168.7	297344	297202.7	297402	**297381.4**	**297456**
8	1500	0.52	302495.7	302613	302568.1	**302792**	**302670.8**	302711
9	1500	0.53	308096.4	308224	308148.7	308231	**308292.8**	**308440**
10	1500	0.54	314198.3	314277	314177.1	314263	**314328.4**	**314463**
11	2000	0.45	466453.5	466748	466489.7	466603	**466667.2**	**466872**
12	2000	0.46	475573.2	475762	475605.2	475724	**475779.4**	**475882**
13	2000	0.47	485711.8	485802	485858.8	486102	**485986.6**	**486122**
14	2000	0.48	495714.7	495772	495737.6	495827	**495940.5**	**496075**
15	2000	0.49	505281.9	505449	505363.5	505565	**505466.2**	**505687**
16	2000	0.50	515741.1	515807	515737.7	515919	**515892.7**	**516056**
17	2000	0.51	526140.2	526320	526128.6	526285	**526280.7**	**526382**
18	2000	0.52	535987.2	536151	536066.9	536221	**536278.2**	**536502**
19	2000	0.53	545706.9	545851	545746.8	**545976**	**545853.0**	545964
20	2000	0.54	556148.2	**556475**	556136.1	556238	**556298.3**	556474
21	2500	0.45	725663.7	725809	725692.0	725783	**725898.6**	**726034**
22	2500	0.46	740798.9	740860	740819.7	741013	**740993.8**	**741058**
23	2500	0.47	756634.5	756703	756705.9	**756997**	**756765.7**	756969
24	2500	0.48	771467.9	771665	771549.8	771732	**771679.2**	**771845**
25	2500	0.49	788344.5	788511	788364.1	788536	**788547.9**	**788818**
26	2500	0.50	803207.0	803526	803157.3	803322	**803275.9**	**803572**
27	2500	0.51	818493.9	818778	818460.7	818623	**818571.5**	**818791**
28	2500	0.52	834306.3	834597	834334.9	834421	**834508.4**	**834836**
29	2500	0.53	849186.7	849358	849241.3	849420	**849404.1**	**849532**
30	2500	0.54	865441.2	865644	865536.8	865694	**865745.8**	**866126**

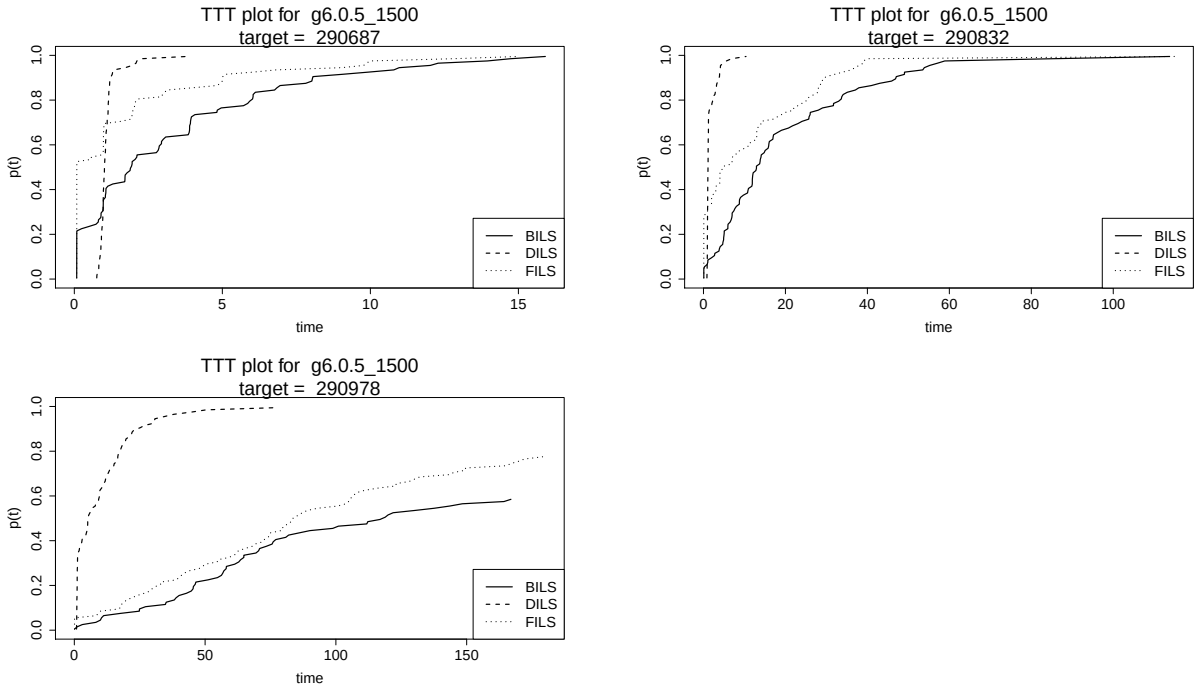

Figura 6.5: O TTT traça um gráfico com 1500 vértices e densidade de 0,5, com o alvo mais fácil no canto superior esquerdo e o alvo mais difícil no canto inferior esquerdo

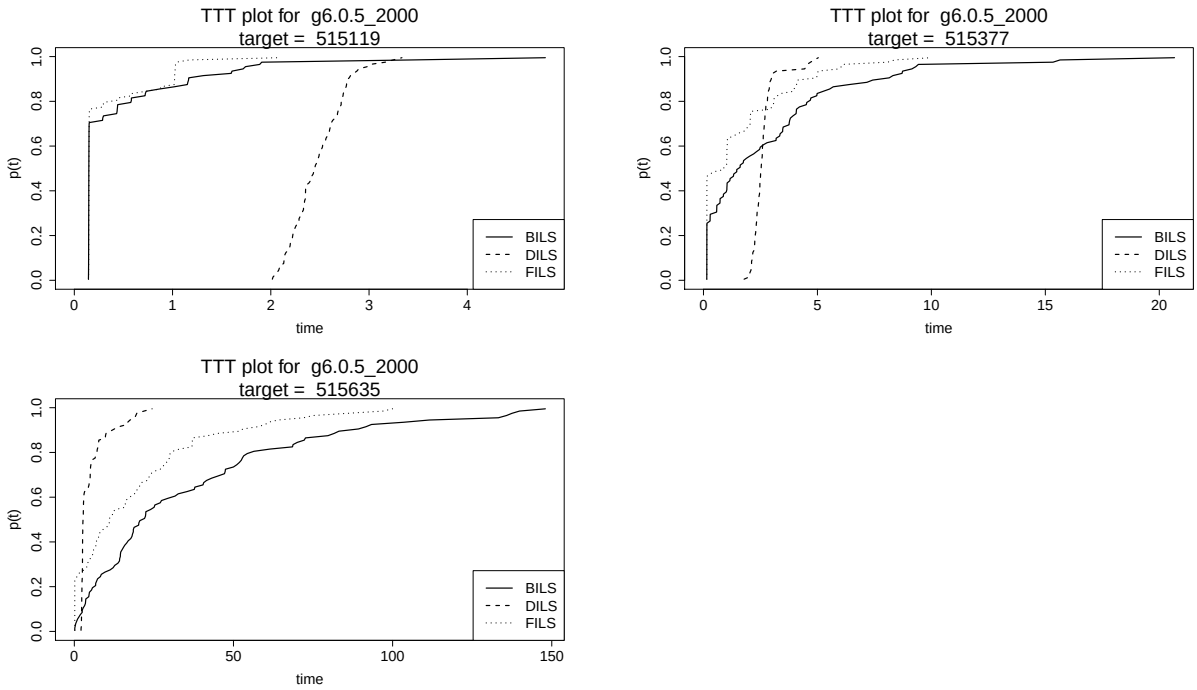

Figura 6.6: O TTT traça um gráfico com 2.000 vértices e densidade de 0,5, com o alvo mais fácil no canto superior esquerdo e o alvo mais difícil no canto inferior esquerdo

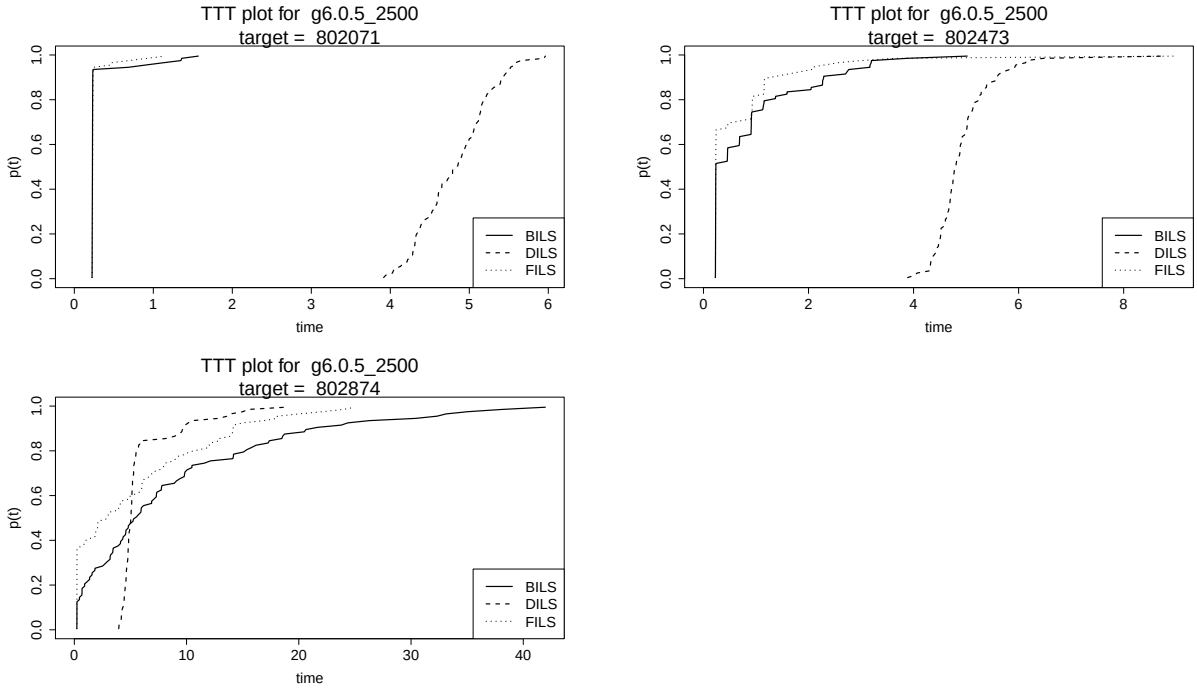

Figura 6.7: O TTT traça um gráfico com 2.500 vértices e densidade de 0,5, com o alvo mais fácil no canto superior esquerdo e o alvo mais difícil no canto inferior esquerdo

Capítulo 7

Conclusão

Nesse trabalho, introduzimos a estratégia do aprimoramento adiado *Delayed Improvement Local Search (DILS)*, uma técnica de busca local baseada na ideia de selecionar como a próxima solução o vizinho aprimorante que tem menos características de soluções ótimas locais. Para implementar essa ideia, avaliamos, para cada solução vizinha, a conformidade com as desigualdades de busca local que são sempre satisfeitas por soluções ótimas locais.

Cada desigualdade dá origem a uma precondição de existência de ótimo local. Há uma correspondência de um para um entre uma precondição violada e uma desigualdade de pesquisa local insatisfeita. Em cada iteração do DILS, o número de verificadores de precondição de existência de ótimo local avaliados como falsos (precondições violadas) é calculado para cada vizinho e esse número, junto ao custo da solução, é usado para determinar a próxima solução corrente.

Implementamos DILS para a busca local 2-opt do problema do caixeiro viajante (TSP) e para a busca local 1-flip do problema corte máximo usando as desigualdades de busca local de [40]. Os resultados computacionais mostram que, para as instâncias testadas, o DILS tem um desempenho melhor do que as implementações clássicas de busca local conhecidas como busca local melhor aprimorante e busca local primeira aprimorante quando a solução inicial é construída com heurísticas gulosas.

A eficácia do DILS depende da disponibilidade do conhecimento de precondições de otimalidade local para os operadores de busca local. Se as desigualdades de busca local não estão disponíveis atualmente para um operador de busca local, precondições de otimalidade local podem ser obtidas estudando as propriedades dos ótimos locais para a vizinhança que está sendo usada.

O desempenho do DILS também depende da implementação do algoritimo. Nesse trabalho, armazenamos informações sobre as soluções vizinhas em iterações anteriores, o que nos ajudou a recalcular rapidamente o número de precondições violadas para cada vizinho após cada etapa da busca local. Com melhores implementações, usando mais informações armazenadas e estruturas de dados mais complexas, pode ser possível melhorar ainda mais o desempenho do DILS.

Pesquisas futuras incluem a aplicação de DILS a outros tipos de vizinhaça para o TSP, como a *SWAP*, para a qual [40] forneceu desigualdades de busca local, e 3-opt.

O DILS também pode ser aplicado a outros problemas de otimização combinatória, além do TSP e do corte máximo. Finalmente, a ideia de DILS ao usar desigualdades de busca local pode ser estendida para metaheurísticas baseadas em busca local, como busca tabu, *Iterated Local Search* e busca de vizinhança variável *(VLS)*. Em cada uma delas pode-se tentar adiar a convergência para um ótimo local, o que pode ser feito usando componentes de busca padrão. Tal implementação pode trazer muitos benefícios.

Referências Bibliográficas

[1] Emile Aarts and Jan K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[2] Renata M Aiex, Mauricio GC Resende, and Celso C Ribeiro. Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366, 2007.

[3] Ahmad Alhindi and Qingfu Zhang. Moea/d with guided local search: Some preliminary experimental results. In *2013 5th Computer Science and Electronic Engineering Conference (CEEC)*, pages 109–114. IEEE, 2013.

[4] Abdullah Alsheddy, Christos Voudouris, Edward P. K. Tsang, and Ahmad Alhindi. *Guided Local Search*, pages 261–297. Springer International Publishing, Cham, 2018.

[5] Heber F Amaral, Sebastián Urrutia, and Lars M Hvattum. Delayed improvement local search. *Journal of Heuristics*, 27(5):923–950, 2021.

[6] Jose-Luis Ambite. Local search in combinatorial optimization. https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume15/ambite01a-html/node9.html, 2001.

[7] David Applegate, William Cook, and André Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[8] Florian Arnold and Kenneth Sörensen. Knowledge-guided local search for the vehicle routing problem. *Computers & Operations Research*, 105:32–46, 2019.

[9] Hüsamettin Bayram and Ramazan Şahin. A new simulated annealing approach for travelling salesman problem. *Mathematical and computational Applications*, 18(3):313–322, 2013.

[10] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Citeseer, 1976.

[11] Rubén Carrasco, Anthanh Pham, Micael Gallego, Francisco Gortázar, Rafael Martí, and Abraham Duarte. Tabu search for the max–mean dispersion problem. *Knowledge-Based Systems*, 85:256–264, 2015.

[12] J Arturo Castillo-Salazar, Dario Landa-Silva, and Rong Qu. Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, 239(1):39–67, 2016.

[13] Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

[14] Sam Cramer and Michael Kampouridis. Optimising the deployment of fibre optics using guided local search. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 799–806. IEEE, 2015.

[15] Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[16] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[17] Daniel Delahaye, Supatcha Chaimatanan, and Marcel Mongeau. Simulated annealing: From basics to applications. In *Handbook of Metaheuristics*, pages 1–35. Springer, 2019.

[18] Abraham Duarte, Jesús Sánchez-Oro, Nenad Mladenović, and Raca Todosijević. Variable neighborhood descent. *Handbook of Heuristics*, pages 341–367, 2018.

[19] Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the tsp. *Algorithmica*, 68(1):190–264, Jan 2014.

[20] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the Max-Cut problem. *Optimization Methods and Software*, 7:1033–1058, 2002.

[21] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[22] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[23] Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision sciences*, 8(1):156–166, 1977.

[24] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[25] Rosmalina Hanafi and Erhan Kozan. A hybrid constructive heuristic and simulated annealing for railway crew scheduling. *Computers & Industrial Engineering*, 70:11–19, 2014.

[26] P Hansen, N Mladenović, Pierre Hansen, Nenad Mladenović, and Les Cahiers Du Gerad. Variable neighborhood search: Methods and recent applications. In *In Proceedings of MIC'99*. Citeseer, 1999.

[27] Pierre Hansen and Nenad Mladenović. Variable neighborhood search: Principles and applications. *European journal of operational research*, 130(3):449–467, 2001.

[28] Pierre Hansen and Nenad Mladenović. First vs. best improvement: an empirical study. *Discrete Applied Mathematics*, 154(5):802–817, 2006.

[29] Pierre Hansen, Nenad Mladenović, Jack Brimberg, and José A. Moreno Pérez. *Variable Neighborhood Search*, pages 57–97. Springer International Publishing, Cham, 2019.

[30] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[31] Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[32] Arianit Islami, Supatcha Chaimatanan, and Daniel Delahaye. Large-scale 4d trajectory planning. In *Air Traffic Management and Systems II*, pages 27–47. Springer, 2017.

[33] David S Johnson, Cecilia R Aragon, Lyle A McGeoch, and Catherine Schevon. Optimization by simulated annealing: an experimental evaluation; part i, graph partitioning. *Operations research*, 37(6):865–892, 1989.

[34] David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of computer and system sciences*, 37(1):79–100, 1988.

[35] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[36] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[37] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[38] Marvin Künnemann and Bodo Manthey. On the smoothed approximation ratio of the 2-opt heuristic for the tsp. In *CTW*, pages 1–4. Citeseer, 2015.

[39] Manuel Laguna. Tabu search. *Handbook of Heuristics*, pages 741–758, 2018.

[40] Giuseppe Lancia, Franca Rinaldi, and Paolo Serafini. Local search inequalities. *Discrete Optimization*, 16:76–89, 2015.

[41] Eugene L Lawler. The traveling salesman problem: a guided tour of combinatorial optimization. *Wiley-Interscience Series in Discrete Mathematics*, 1985.

[42] Xinyu Li and Liang Gao. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics*, 174:93–110, 2016.

[43] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[44] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[45] Zhipeng Lü, Jin-Kao Hao, and Fred Glover. Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics*, 17(2):97–118, 2011.

[46] Wayan Firdaus Mahmudy. Improved simulated annealing for optimization of vehicle routing problem with time windows (vrptw). *Kursor*, 7(3), 2016.

[47] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[48] Temel Öncan, İ Kuban Altınel, and Gilbert Laporte. A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers & Operations Research*, 36(3):637–654, 2009.

[49] Sally Chen Woon Peh and Jer Lang Hong. Glsdock–drug design using guided local search. In *International Conference on Computational Science and Its Applications*, pages 11–21. Springer, 2016.

[50] Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[51] August G Roesener and J Wesley Barnes. An advanced tabu search approach to the dynamic airlift loading problem. *Logistics Research*, 9(1):12, 2016.

[52] August G Roesener, J Wesley Barnes, James T Moore, and David A Van Veldhuizen. An advanced tabu search approach to the static airlift loading problem. *Military Operations Research*, pages 5–29, 2010.

[53] Nasser Tairan, Abdulmohsen Algarni, Justin Varghese, and Muhammad Asif Jan. Population-based guided local search for multidimensional knapsack problem. In *2015*

Fourth International Conference on Future Generation Communication Technology (FGCT), pages 1–5. IEEE, 2015.

[54] Christos Tsotskas, Timoleon Kipouros, and Anthony Mark Savill. The design and implementation of a gpu-enabled multi-objective tabu-search intended for real world and high-dimensional applications. *Procedia Computer Science*, 29:2152–2161, 2014.

[55] Christos Voudouris. *Guided local search for combinatorial optimisation problems.* PhD thesis, University of Essex, 1997.

[56] Grzegorz Waligóra. Simulated annealing and tabu search for discrete-continuous project scheduling with discounted cash flows. *RAIRO-Operations Research*, 48(1):1–24, 2014.

[57] Qinghua Wu, Jin-Kao Hao, and Fred Glover. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196(1):611–634, 2012.

[58] Ivan Žulj, Sergej Kramer, and Michael Schneider. A hybrid of adaptive large neighborhood search and tabu search for the order-batching problem. *European Journal of Operational Research*, 264(2):653–664, 2018.