Uma abordagem contínua para o problema do caixeiro viajante

Paula Cristina Rohr Ertel

Dissertação apresentada ao
Instituto de Matemática e Estatística
da Universidade de São Paulo
para obtenção do título de
Mestre em Ciências

Programa de Matemática Aplicada
Orientador: Ernesto G. Birgin

Durante o desenvolvimento deste trabalho a autora recebeu auxílio financeiro da CAPES

São Paulo, Brasil
Março 2023

Uma abordagem contínua para o problema do caixeiro viajante

Paula Cristina Rohr Ertel

Dissertação apresentada ao Programa de Pós-Graduação em Matemática Aplicada do Instituto de Matemática e Estatística da Universidade de São Paulo como parte dos requisitos para obtenção do título de Mestre em Ciências.

Banca Examinadora:

- Prof. Dr. Ernesto G. Birgin (orientador) - IME-USP
- Prof. Dr. José Mario Martínez - IMECC-UNICAMP
- Prof. Dr. Luiz Rafael dos Santos - UFSC

Ficha catalográfica elaborada com dados inseridos pelo(a) autor(a)
Biblioteca Carlos Benjamin de Lyra
Instituto de Matemática e Estatística
Universidade de São Paulo

Ertel, Paula Cristina Rohr
Uma abordagem contínua para o problema do caixeiro viajante / Paula Cristina Rohr Ertel; orientador, Ernesto Julian Goldberg Birgin. - São Paulo, 2023.
76 p.: il.

Dissertação (Mestrado) - Programa de Pós-Graduação em Matemática Aplicada / Instituto de Matemática e Estatística / Universidade de São Paulo.
Bibliografia
Versão original

1. Problema do Caixeiro Viajante. 2. Problema do Caixeiro Viajante Contínuo. 3. Heurísticas. 4. Métodos de Busca em Bloco de Coordenadas. I. Birgin, Ernesto Julian Goldberg. II. Título.

Bibliotecárias do Serviço de Informação e Biblioteca
Carlos Benjamin de Lyra do IME-USP, responsáveis pela
estrutura de catalogação da publicação de acordo com a AACR2:
Maria Lúcia Ribeiro CRB-8/2766; Stela do Nascimento Madruga CRB 8/7534.

Resumo

O problema do caixeiro viajante contínuo ou CTSP, do inglês *Continuous Traveling Salesman Problem*, é uma variante do problema do caixeiro viajante (TSP) na qual cada cidade pertence a um conjunto arbitrário e o objetivo é determinar o tour de menor comprimento que passa por cada um dos conjuntos e retorna ao primeiro. Neste trabalho consideramos o CTSP em que os conjuntos visitados são dados por bolas pertencentes ao $\mathbb{R}^2$. Ou seja, dado um conjunto de m bolas do $\mathbb{R}^2$, estamos interessados em determinar pontos x^i, um em cada bola, e uma permutação de modo que o tour passando pelos pontos x^i com a ordem dada pela permutação tenha comprimento mínimo. Para resolver este problema propomos quatro algoritmos, nos quais unimos heurísticas já consolidadas para o TSP clássico, como a heurística 2-Opt, com um método de busca em bloco de coordenadas. Os algoritmos foram implementados na linguagem de programação `Fortran 90` e para testá-los geramos 60 instâncias de pontos aleatórios, sendo 10 para cada número de bolas $m \in \{50, 100, 200, 300, 400, 500\}$. Além disso, foram realizados experimentos numéricos com 10 instâncias da biblioteca TSPLIB adaptadas ao CTSP. Tabelas foram geradas para cada experimento, detalhando as soluções encontradas pelos métodos e seus desempenhos, permitindo realizar comparações entre eles. Figuras das soluções encontradas pelos métodos também são apresentadas para algumas instâncias.

Palavras-chave: Problema do Caixeiro Viajante, Problema do Caixeiro Viajante Contínuo, Heurísticas, Métodos de Busca em Bloco de Coordenadas.

Abstract

The Continuous Traveling Salesman Problem (CTSP) is a variant of the Traveling Salesman Problem (TSP) in which each city belongs to an arbitrary set and the aim is to determine the shortest tour passing through each of the sets and returning to the first. In this work we consider the CTSP in which the sets visited are given by balls in $\mathbb{R}^2$. That is, given a set of m balls in $\mathbb{R}^2$, we are interested in determining points x^i, one on each ball, and a permutation such that the tour among the balls that visit each point x^i exactly once with the order given by the permutation has minimum length. To solve this problem we propose four algorithms, in which we combine consolidated heuristics for the classical TSP, such as the 2-Opt heuristic, with a block coordinate descent method. The algorithms were implemented with `Fortran 90` programming language and to test them we generated 60 instances of random points, being 10 for each number of balls $m \in \{50, 100, 200, 300, 400, 500\}$. In addition, numerical experiments were performed with 10 instances of the TSPLIB library adapted to CTSP. Tables were generated for each experiment, detailing the solutions obtained by the methods and their performances, allowing comparisons among them. Figures of the solutions obtained by the methods are also presented for some instances.

Keywords: Traveling Salesman Problem, Continuous Traveling Salesman Problem, Heuristics, Block Coordinate Descent Methods.

Sumário

Lista de figuras **7**

Lista de tabelas **8**

1 Introdução **9**

2 Problema do caixeiro viajante **11**

2.1 Heurísticas para o problema do caixeiro viajante 13

2.1.1 Heurísticas construtivas 14

2.1.2 Heurísticas de busca local 16

3 Problema do caixeiro viajante contínuo **19**

3.1 Definição do problema 19

3.2 Métodos de busca em coordenadas 21

3.3 Algoritmos para o TSP contínuo 24

4 Experimentos Numéricos **36**

4.1 Experimentos iniciais 36

4.2 Experimentos com instâncias aleatórias 41

4.3 Experimentos com instâncias da biblioteca TSPLIB 56

5 Considerações finais **66**

Bibliografia **68**

Apêndice A **71**

Lista de Figuras

2.1 Campanha publicitária da empresa *Procter & Gamble* com um problema do caixeiro viajante de 33 cidades. Imagem retirada de [2]. 12
2.2 A heurística NN não necessariamente gera tours ótimos. 15
2.3 O ponto inicial pode influenciar no tour gerado pela heurística NN. 15
2.4 Exemplo de movimento 2-Opt. Fonte: [1] 18
2.5 Exemplo de movimento 3-Opt. Fonte: [1] 18

3.1 Exemplo ilustrativo de pontos viáveis x^{i_ν}. 26
3.2 A figura da esquerda ilustra um exemplo no qual o problema (3.11) admite infinitas soluções, enquanto que na figura da direita temos um exemplo em que a solução é única. 27

4.1 Evolução dos tours a cada ciclo do método BDC (3). 37
4.2 Evolução dos tours a cada ciclo do Algoritmo 3 para uma instância de 24 bolas com raios uniformes. 38
4.3 Construção do tour inicial pelo Algoritmo 4 para a instância com 40 bolas. 39
4.4 Amostra de tours obtidos pelo método 2optBest para a instância de 40 bolas. . . . 40
4.5 Exemplos de instâncias para o CTSP. 41
4.6 Gráficos do comprimento do tour em função do tempo, em segundos, obtidos pelos métodos quando aplicados às instâncias com 100 bolas. 44
4.7 Comprimento do tour em função do tempo de cada método para a instância I2 de 100 bolas. 45
4.8 Construção do tour inicial para a instância I2 de 100 bolas. 46
4.9 Evolução dos tours a cada iteração do método 2optBest. 47
4.10 Solução encontrada pelo método 2optBest para a instância I2. Iteração: 29. Comprimento: 62,87 u.c. Tempo de execução: 17,6s. 47
4.11 Evolução dos tours a cada iteração do método 2optFirst. 48
4.12 Solução encontrada pelo método 2optFirst para a instância I2. Iteração: 66. Comprimento: 64,26 u.c. Tempo de execução: 9,8s. 49
4.13 Evolução dos tours a cada iteração do método InsercaoBest. 50
4.14 Solução encontrada pelo método InsercaoBest para a instância I2. Iteração: 42. Comprimento: 66,20 u.c. Tempo de execução: 52,42s. 50

4.15 Evolução dos tours a cada iteração do método InsercaoFirst. 51
4.16 Solução encontrada pelo método InsercaoFirst para a instância I2. Iteração: 103. Comprimento: 67,29 u.c. Tempo de execução: 47,75s. 52
4.17 Gráficos das iterações e do tempo, respectivamente, em função do número de bolas. 54
4.18 Construção do tour inicial para a instância I9 de 500 bolas. 55
4.19 Soluções encontradas por cada um dos métodos para a instância I9 com 500 bolas. 56
4.20 Exemplos de instâncias da biblioteca TSPLIB adaptadas ao CTSP. 57
4.21 Gráficos do comprimento do tour em função do tempo. 60
4.22 Soluções encontradas por cada um dos métodos para a instância *ch150*. 62
4.23 Soluções encontradas por cada um dos métodos para a instância *pr264*. 63
4.24 Soluções encontradas por cada um dos métodos para a instância *d493*. 64

Lista de Tabelas

4.1 Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 100$ bolas. 43
4.2 Desempenho do método 2optBest (5) em instâncias aleatórias. 52
4.3 Desempenho do método 2optFirst (6) em instâncias aleatórias. 53
4.4 Desempenho do método InsercaoBest (7) em instâncias aleatórias. 53
4.5 Desempenho do método InsercaoFirst (8) em instâncias aleatórias. 53
4.6 Desempenho do método 2optBest (5) em instâncias da biblioteca TSPLIB. 58
4.7 Desempenho do método 2optFirst (6) em instâncias da biblioteca TSPLIB. 59
4.8 Desempenho do método InsercaoBest (7) em instâncias da biblioteca TSPLIB. . . 59
4.9 Desempenho do método InsercaoFirst (8) em instâncias da biblioteca TSPLIB. . . 59
4.10 Comprimento da solução ótima conhecida para cada instância considerando o TSP clássico e das soluções encontradas pelos algoritmos para o CTSP. 61

5.1 Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 50$ bolas. 72
5.2 Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 200$ bolas. 73
5.3 Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 300$ bolas. 74
5.4 Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 400$ bolas. 75
5.5 Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 500$ bolas. 76

Capítulo 1

Introdução

O problema do caixeiro viajante (ou TSP, do inglês *Traveling Salesman Problem*) está entre os problemas mais estudados de otimização combinatória. Tal problema consiste na seguinte questão: dado um conjunto de m cidades, com a distância de se viajar de uma cidade i para qualquer outra cidade j conhecida, desejamos encontrar o tour de menor comprimento que passe por todas as m cidades uma única vez e retorne à cidade inicial. Apesar de ser um problema simples de descrever, o TSP pertence à classe de problemas NP-difícil [9] e, junto a suas variações, possui aplicações nas mais diversas áreas do conhecimento, como logística, genética, neurociência, entre muitas outras [2].

Apesar do TSP ser um problema que já foi intensivamente investigado, seu estudo ainda apresenta muitas possibilidades, sobretudo quando consideramos novas abordagens para o problema. Uma delas é o problema do caixeiro viajante contínuo (ou CTSP, do inglês *Continuous Traveling Salesman Problem*) proposto em [6]. O CTSP consiste em: dado um conjunto de m polígonos Ω_i, desejamos encontrar pontos $x^i \in \Omega_i$ para $i = 1, \ldots, m$ e uma permutação i_1, i_2, $\ldots$, i_m que minimize a função

$$f(i_1, \ldots, i_m; x) \coloneqq \|x^{i_n} - x^{i_1}\| + \sum_{\nu=1}^{n-1} \|x^{i_\nu} - x^{i_{\nu+1}}\|, \tag{1.1}$$

em que $x^{i_\nu} = (x_1^{i_\nu}, x_2^{i_\nu}) \in \mathbb{R}^2$ para $\nu = 1, \ldots, m$, $x^T = ((x^1)^T, \ldots, (x^m)^T) \in \mathbb{R}^{2m}$ e $\|\cdot\|$ corresponde à norma Euclidiana. Neste trabalho vamos considerar bolas $\mathcal{B}(c^i, r^i) \in \mathbb{R}^2$ no lugar dos polígonos Ω_i. Para resolver o problema (1.1) propomos unir heurísticas já consolidadas para o TSP clássico com um método de otimização contínua: o método de busca em bloco de coordenadas.

A dissertação está estruturada da seguinte forma: no Capítulo 2 introduzimos a formulação clássica do problema do caixeiro viajante e as heurísticas utilizadas nos experimentos numéricos. Em seguida, no Capítulo 3, apresentamos a formulação do problema (1.1) e comentamos sua conexão com outra variante do TSP conhecida como CETSP. Também desenvolvemos brevemente a teoria dos métodos de busca em coordenadas e detalhamos os pseudocódigos dos métodos imple-

mentados. Os experimentos numéricos são apresentados no Capítulo 4. Em todos os experimentos utilizamos instâncias do $\mathbb{R}^2$ e trabalhamos com a distância Euclidiana entre eles para obter e avaliar o comprimento dos tours gerados. Os experimentos foram divididos em três grupos. Inicialmente, na Seção 4.1, testamos os algoritmos propostos no Capítulo 3 em instâncias construídas com a intenção de possibilitar a visualização da atuação dos métodos. Para o segundo grupo de experimentos, conjuntos de instâncias aleatórias foram gerados considerando diferentes quantidades de bolas. Por fim, o terceiro grupo de testes, abordado na Seção 4.3, foi realizado com 10 instâncias da biblioteca TSPLIB [28] adaptadas ao CTSP. O Capítulo 5 conclui este trabalho com algumas considerações finais e propostas de trabalhos futuros.

Capítulo 2

Problema do caixeiro viajante

Com uma lista de m cidades em mãos, um caixeiro viajante deseja visitar cada uma dessas cidades exatamente uma vez e então retornar a primeira cidade visitada. As distâncias de uma cidade i para qualquer outra cidade j são conhecidas. Com tais informações, o caixeiro levanta a seguinte questão [15]: qual o tour de menor comprimento possível que ele pode tomar? Aqui, *tour* se refere à ordem com a qual as cidades devem ser visitadas e o *comprimento do tour* é dado pela soma das distâncias entre elas.

A princípio a pergunta levantada pelo caixeiro pode parecer simples, e com um lápis e papel em mãos alguém pode prontamente, e ingenuamente, se propor a resolvê-lo escrevendo todos os tours possíveis na busca por aquele de comprimento mínimo. No entanto, mesmo exemplos pequenos, com poucas cidades, já demonstram que tal abordagem é inviável. Em 1962, a companhia americana *Procter and Gamble* ofereceu, em uma campanha publicitária, um prêmio de US$10.000,00 para quem encontrasse a rota mais curta partindo de Chicago e passando pelas 32 cidades em vermelho na Figura 2.1. Para resolver esse problema com a estratégia anteriormente mencionada, de avaliar todos os tours possíveis, seria necessário avaliar 32! tours, isto é, mais de 2.63×10^{35} tours.

O problema do caixeiro viajante (TSP, do inglês *Traveling Salesman Problem* [2, 8, 18, 16, 17]) introduzido no primeiro parágrafo é um dos mais famosos e estudados problemas da área de otimização combinatória. A origem de seu nome é incerta, aparecendo pela primeira vez para se referir ao problema em questão em um relatório de 1949 de Julia Robison intitulado “On the Hamiltonian game (a traveling salesman problem)” [29]. No entanto, de acordo com Applegate et al. [2], a autora não parece utilizar o termo como sendo de sua autoria e conclui que o nome tenha surgido em algum momento entre 1930 e 1940, em Princeton, momento a partir do qual ele se torna objeto de amplo estudo.

Matematicamente, na versão clássica do problema do caixeiro viajante é fornecido um conjunto de m cidades $\{c_1, c_2, \ldots, c_m\}$ e uma distância $d(c_i, c_j)$ para cada par $\{c_i, c_j\}$ de cidades distintas.

Figura 2.1: Campanha publicitária da empresa *Procter & Gamble* com um problema do caixeiro viajante de 33 cidades. Imagem retirada de [2].

O objetivo é encontrar uma ordem π das cidades que minimize

$$d(c_{\pi(m)}, c_{\pi(1)}) + \sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}). \tag{2.1}$$

A soma (2.1) corresponde ao *comprimento do tour* que um caixeiro faria ao visitar cada uma das cidades na ordem dada pela permutação π e retornar ao final para a primeira cidade visitada.

O TSP também pode ser formulado em termos da teoria de grafos. Seja $G_m(V, E)$ um grafo completo e ponderado, em que $V = \{1, \ldots, m\}$ denota o conjunto de vértices (cidades ou clientes) e E é o conjunto de arestas, com pesos w_{ij}, conectando os vértices. Neste contexto, o TSP consiste

em determinar o *ciclo Hamiltoniano* $\mathcal{C}$ que minimiza a função

$$f(\mathcal{C}) := \sum_{\{ij\}\in\mathcal{C}} w_{ij}, \tag{2.2}$$

em que os pesos w_{ij} podem ser definidos como as distâncias Euclidianas entre as cidades i e j. Um ciclo Hamiltoniano de um grafo não orientado $G(V, E)$ é um ciclo simples que contém todos os vértices de V.

A versão *simétrica* do problema do caixeiro viajante (STSP) corresponde ao caso em que $d(c_i, c_j) = d(c_j, c_i)$ para todos $1 \leq i, j \leq m$. O TSP *geométrico* se refere às instâncias em que as cidades são localizações e o custo de se viajar de uma cidade a outra é dado pela distância entre elas. No TSP Euclidiano (ou *métrico*), que pode ser visto como um caso particular do geométrico, cada nó (ou cidade) corresponde a um vetor no espaço Euclidiano e a distância entre cada par de nós é dada pela norma (Euclidiana) da diferença entre seus vetores.

2.1 Heurísticas para o problema do caixeiro viajante

Como mencionado anteriormente, o problema do caixeiro viajante pertence à classe de problemas NP-difícil (uma demonstração deste resultado pode ser encontrada em Cormen et al. [9]). Isso significa que a probabilidade de encontrar um algoritmo de tempo polinomial para resolver este problema com exatidão é baixa. Neste contexto, uma alternativa é a busca por soluções aproximadas. Algoritmos de aproximação [1] sacrificam a otimalidade em prol da eficiência, produzindo soluções "quase" ótimas, isto é, soluções cujo valor não é pior que uma fração pré-determinada do ótimo, dentro de um tempo computacional razoável (polinomial). Entretanto, para alguns problemas, obter algoritmos de aproximação também pode ser tão difícil quanto obter algoritmos exatos. Este é caso do TSP, para o qual apenas algumas formulações particulares admitem algoritmos de aproximação, como por exemplo o algoritmo de Christofides [1, 7] para o caixeiro viajante métrico. Assim, novas estratégias são necessárias e é neste contexto que os métodos heurísticos ocupam um lugar de destaque.

As heurísticas diferem dos algoritmos de aproximação no sentido de que não necessariamente possuem uma garantia de qualidade da solução encontrada, isto é, uma relação do quão longe ela está do valor ótimo do problema. A literatura sobre métodos heurísticos para o problema do caixeiro viajante é vasta e segue em desenvolvimento. Tais métodos podem ser projetados tendo em vista diferentes objetivos, como velocidade, construção ou a melhora de tours já existentes. Vamos nos ater aqui às duas principais classes: os métodos construtivos e os métodos de busca local.

2.1.1 Heurísticas construtivas

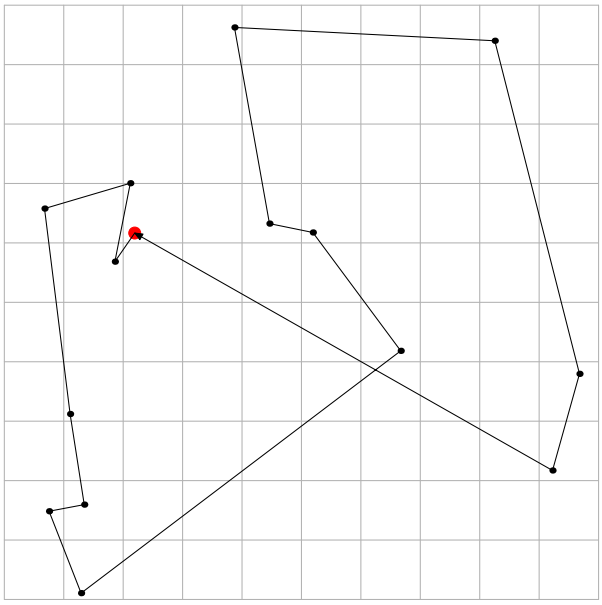
Heurísticas construtivas, como o próprio nome sugere, são algoritmos que atuam construindo um tour de maneira gradual até que um tour válido seja criado. Alguns exemplos são a heurística do vizinho mais próximo (ou NN, do inglês *Nearest Neighbor*), o algoritmo de Christofides [7] e a heurística construtiva de inserção [13], que a depender da regra de construção recebe diferentes nomes: *nearest insertion*, *cheapest insertion* ou *farthest insertion*.

Tendo em vista os experimentos numéricos a serem realizados, descrevemos a seguir apenas a heurística do vizinho mais próximo, pois ela será utilizada para construir o tour inicial para as heurísticas de busca local. O leitor interessado pode se referir a Aarts and Lenstra [1], Capítulo 8, para uma descrição das outras heurísticas construtivas mencionadas.

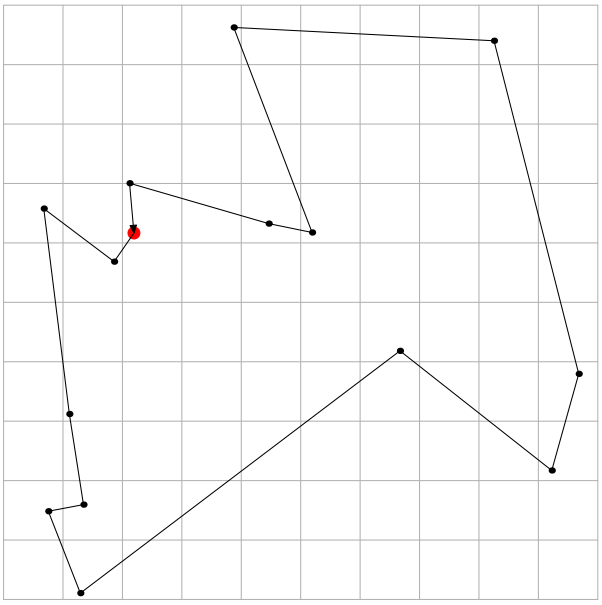
Heurística do vizinho mais próximo (NN)

Na heurística do vizinho mais próximo o caixeiro inicia seu tour em uma cidade, a princípio qualquer, e segue para a cidade ainda não visitada mais próxima da atual até que todas as cidades tenham sido visitadas. Em outras palavras, para construir uma ordem de visitação π do tour, primeiro uma cidade inicial $c_{\pi(1)}$ é escolhida arbitrariamente. Em seguida, para i de 1 até $m-1$, $c_{\pi(i+1)}$ recebe a cidade c_k que minimiza $\{d(c_{\pi(i)}, c_k) : k \neq \pi(j), 1 \leq j \leq i\}$. Por fim, adicionamos a aresta $\{c_{\pi(m)}, c_{\pi(1)}\}$ completando o tour.

A complexidade computacional desta heurística é $O(m^2)$. Apesar de se constituir numa abordagem muito natural para a construção de soluções viáveis para o TSP, a heurística NN não fornece, em geral, soluções ótimas. Um dos fatores que prejudica a qualidade de soluções geradas com NN é o fato de que algumas cidades são "esquecidas" durante o processo e acabam sendo inseridas com um grande custo ao final. Na Figura 2.2 podemos visualizar isso acontecendo. Neste exemplo as cidades são pontos do plano e a distância entre elas é dada pela distância Euclidiana. O ponto em vermelho representa a cidade inicial e a flecha a orientação do tour. As Figuras 2.2a e 2.2b correspondem a dois tours distintos para um mesmo conjunto de pontos. A Figura 2.2a corresponde ao tour gerado utilizando a heurística NN, enquanto que a Figura 2.2b apresenta o tour obtido a partir de uma heurística de busca local. O comprimento de cada tour é apresentado junto da respectiva figura, com a notação "u.c." denotando "unidades de comprimento".



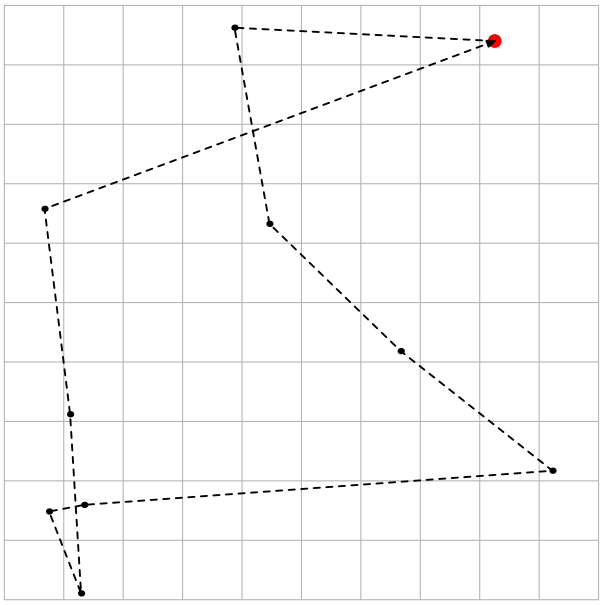
(a) Comprimento do tour: $43,81$ u.c.



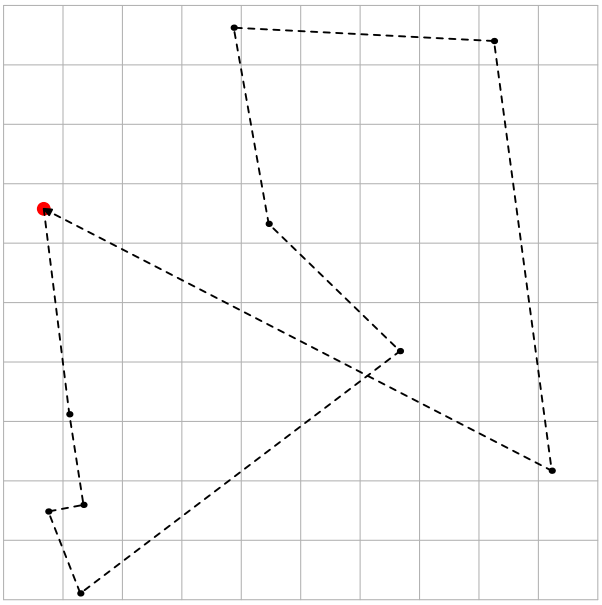
(b) Comprimento do tour: $38,75$ u.c.

Figura 2.2: A heurística NN não necessariamente gera tours ótimos.

Outro detalhe que deve ser considerado sobre a heurística do vizinho mais próximo é o fato de a solução gerada por ela depender da cidade inicial. Ou seja, para um mesmo conjunto, se o método for iniciado com diferentes cidades ele poderá retornar soluções distintas. Na Figura 2.3 por exemplo, os pontos em vermelho representam as cidades iniciais, e observe que as soluções obtidas pela heurística NN foram diferentes, apesar de o conjunto de pontos ser o mesmo.



(a) Comprimento do tour: $38,64$ u.c.



(b) Comprimento do tour: $41,58$ u.c.

Figura 2.3: O ponto inicial pode influenciar no tour gerado pela heurística NN.

2.1.2 Heurísticas de busca local

Heurísticas de busca local atuam de maneira iterativa, modificando e melhorando uma dada solução inicial. Por isso, também podem ser chamadas de heurísticas de melhoria. Em tais métodos a noção de vizinhança é imprescindível e, portanto, apresentamos inicialmente algumas definições importantes neste contexto para em seguida descrever alguns métodos que pertencem a esta classe. As definições a seguir seguem o livro de Aarts and Lenstra [1].

Em problemas de otimização combinatória, classe a qual pertence o TSP, é comum falarmos em *instâncias* para o problema.

Definição 1. Uma *instância* de um problema de otimização combinatória é um par $(\mathcal{I}, f)$, em que $\mathcal{I}$ é o conjunto de soluções viáveis e f corresponde à função custo $f : \mathcal{I} \to \mathbb{R}$. O problema consiste em encontrar uma solução ótima global, isto é, $i^* \in \mathcal{I}$ tal que $f(i^*) \leq f(i)$ para todo $i \in \mathcal{I}$. Além disso, $f^* := f(i^*)$ denota o custo ótimo e $\mathcal{I}^* = \{i \in \mathcal{I}; f(i) = f^*\}$ denota o conjunto de soluções ótimas.

No contexto do problema do caixeiro viajante, o conjunto de soluções viáveis $\mathcal{I}$ é dado pelo conjunto de tours e f é a função objetivo que retorna o custo total de um tour, isto é, seu comprimento. Ou, pensando na formulação do TSP em termos da teoria de grafos (2.2), $\mathcal{I}$ pode ser escolhido como o conjunto de todos os ciclos Hamiltonianos $\mathcal{C}$ do grafo G_m.

Definição 2. Seja $(\mathcal{I}, f)$ uma instância de um problema de otimização combinatória. Uma função de vizinhança $\mathcal{N}$ é um mapa que define para cada solução $i \in \mathcal{I}$ um conjunto $\mathcal{N}(i) \subseteq \mathcal{I}$ de soluções que são, em algum sentido, próximas de i. O conjunto $\mathcal{N}(i)$ é a *vizinhança* de uma solução i e cada $j \in \mathcal{N}(i)$ é um *vizinho* de i. Vamos assumir que $i \in \mathcal{N}(i)$ para todo $i \in \mathcal{I}$.

Portanto, um algoritmo de busca local é constituído de três etapas básicas: (i) gerar uma solução inicial; (ii) gerar uma solução vizinha; (iii) calcular o custo de soluções. As diferentes formas de se gerar a vizinhança é que originam os diferentes métodos de busca local. Assim, uma busca local consiste num processo de melhoria iterativa, que inicia com uma solução viável (obtida com alguma heurística construtiva) e busca em sua vizinhança por uma solução com custo menor. Se ela é encontrada, substitui a solução atual e a busca continua. Caso contrário, o algoritmo retorna a solução atual, que será um ótimo local de acordo com a definição a seguir.

Definição 3. Seja $(\mathcal{I}, f)$ uma instância de um problema de otimização combinatória e $\mathcal{N}$ uma função de vizinhança. Uma solução viável $\hat{\imath} \in \mathcal{I}$ é um *ótimo local* com respeito a $\mathcal{N}$ se $f(\hat{\imath}) \leq f(i)$ para todo $i \in \mathcal{N}(\hat{\imath})$. Denotamos o conjunto de ótimos locais por $\hat{\mathcal{I}}$.

Observe que a otimalidade local depende da função de vizinhança.

A escolha de um vizinho para atualizar a solução atual e continuar a busca local pode ser feita de acordo com duas estratégias distintas [1]. Na estratégia do *primeiro vizinho* que melhora, a solução atual é substituída pela primeira solução de menor custo encontrada durante uma busca pela vizinhança. A estratégia do *melhor vizinho*, por sua vez, avalia o custo de todos os vizinhos da solução atual e a substitui pela melhor solução dessa vizinhança.

Fornecemos acima um panorama geral de como as heurísticas de busca local atuam. Agora, nosso interesse será detalhar algumas heurísticas específicas para o caixeiro viajante, sobretudo as operações que elas implementam para gerar a vizinhança de um tour a cada iteração e então realizar a busca local. Nos limitaremos às heurísticas utilizadas nos experimentos numéricos: a heurística 2-Opt e a heurística de inserção. Ambas são caracterizadas por fazer "movimentos" que transformam um tour em outro. Desse modo, a vizinhança de um dado tour é composta por todos aqueles que podem ser obtidos através desse movimento pré-especificado.

Heurística de melhoria k-Opt

Como comentado anteriormente, nas heurísticas de busca local projetadas para o problema do caixeiro viajante, uma estrutura de vizinhança é definida no conjunto de tours (soluções viáveis) de modo que um tour i' é dito vizinho do tour i se eles se diferem de uma maneira pré-especificada. Desse modo, as heurísticas k-Opt se caracterizam por definir vizinhanças k-Opt, nas quais um tour i' é obtido a partir de i removendo-se k arestas e as substituindo por um novo conjunto de k arestas diferente. Esta mudança recebe o nome de *movimento k-Opt*.

Assim, as heurísticas 2-Opt e 3-Opt, as mais famosas dentre os algoritmos de busca local, são casos particulares da heurística k-Opt que utilizam movimentos 2-Opt e 3-Opt, respectivamente.

De acordo com Aarts and Lenstra [1], o algoritmo 2-Opt foi inicialmente proposto por Croes [10], mas o movimento que o caracteriza já havia sindo sugerido por Flood [12]. A Figura 2.4 exemplifica a atuação do movimento 2-Opt, o qual remove duas arestas de um dado tour quebrando-o em dois caminhos e então os reconecta de modo a gerar um tour diferente. Sendo m o número de cidades do problema, a complexidade computacional desta heurística é $O(m^2)$.

O movimento 3-Opt, por sua vez, remove três arestas, quebrando o tour em três caminhos que serão reconectados para obter novos tours. A Figura 2.5 exemplifica a atuação do movimento 3-Opt.

Para a heurística 3-Opt, a complexidade computacional é $O(m^3)$.

Heurística de inserção

A heurística de inserção, enquanto um método de busca local, é caracterizada por utilizar um movimento de *realocação* [1, p. 342] das cidades para construir a vizinhança. Este movimento

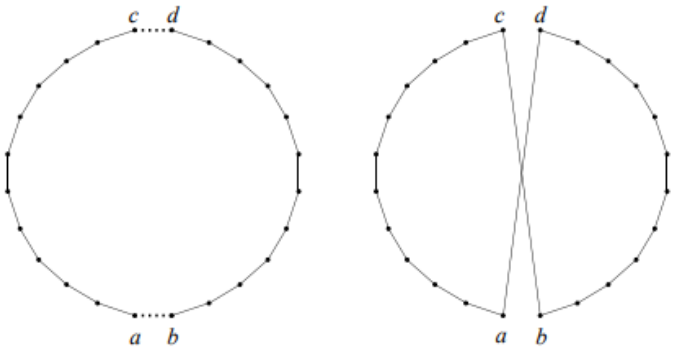


Figura 2.4: Exemplo de movimento 2-Opt. Fonte: [1]

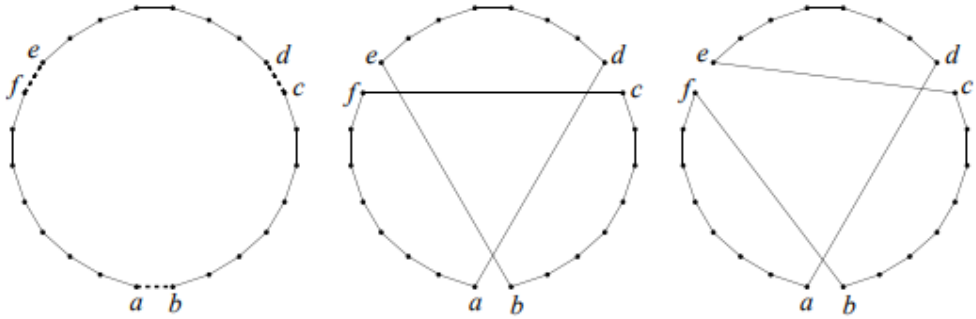


Figura 2.5: Exemplo de movimento 3-Opt. Fonte: [1]

funciona da seguinte forma: dada uma permutação $\pi := (\pi(1), \pi(2), \ldots, \pi(m))$, cada $\pi(i)$, para $i = 1, \ldots, m$, é removido e reinserido em todas as posições possíveis $t \neq i$. Logo, a complexidade computacional desta heurística é $O(m^2)$.

No movimento de realocação clássico, cada cidade é reinserida entre duas cidades consecutivas do tour. No entanto, podemos encontrar na literatura (veja Gendreau et al. [13, 14]) variações da heurística de inserção em que a reinserção pode ser feita entre cidades próximas e não necessariamente consecutivas.

Capítulo 3

Problema do caixeiro viajante contínuo

No capítulo anterior introduzimos a versão clássica do problema do caixeiro viajante (TSP). Nela, as cidades (clientes, pontos, nós, vértices, ...) estão fixas e o objetivo consiste em determinar a ordem com a qual o caixeiro deve visitá-las de modo a obter um tour de comprimento mínimo. Agora, no capítulo atual, pretendemos formular e propor algoritmos para resolver uma variante do TSP Euclidiano proposta em [6]: o problema do caixeiro viajante contínuo (ou CTSP, do inglês *Continuous Traveling Salesman Problem*). No CTSP, cada cidade está contida em um conjunto arbitrário e o problema consiste em determinar o tour de menor comprimento que passa por cada conjunto.

A definição do nosso problema será apresentada a seguir, na Seção 3.1. Na Seção 3.2 abordamos, de forma sucinta, a teoria dos métodos de busca em coordenadas. A Seção 3.3, por sua vez, contêm os algoritmos propostos para a resolução numérica do CTSP.

3.1 Definição do problema

No TSP Euclidiano, dado um conjunto de m pontos com as distâncias Euclidianas $d_{ij} > 0$ entre eles conhecidas, desejamos encontrar uma permutação $i_1, i_2, \ldots, i_m$ que minimiza a soma

$$d_{i_m,i_1} + \sum_{\nu=1}^{m-1} d_{i_\nu,i_{\nu+1}}.$$

Esta soma corresponde ao comprimento total do tour passando pelos m pontos uma única vez e retornando ao primeiro ponto visitado. Na variante que formularemos a seguir os pontos a serem visitados não estão fixados e, portanto, as distâncias podem variar.

Considere um conjunto de m polígonos $\Omega_1, \Omega_2, \ldots, \Omega_m$, que podem ser não convexos. O CTSP,

da forma como é formulado em [6], se concentra em encontrar pontos $x^i \in \Omega_i$ e uma permutação $i_1, i_2, \ldots, i_m$ que minimize o custo

$$\|x^{i_m} - x^{i_1}\| + \sum_{\nu=1}^{m-1} \|x^{i_\nu} - x^{i_{\nu+1}}\|.$$

No entanto, neste trabalho, vamos considerar os conjuntos Ω_i como sendo bolas do $\mathbb{R}^2$. Assim, mais precisamente, dado um conjunto de m bolas $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_m$, tal que para cada $i = 1, \ldots, m$, $\mathcal{B}_i := \mathcal{B}(c^i, r^i) \in \mathbb{R}^2$ e $c^i \in \mathbb{R}^2$ e $r^i \in \mathbb{R}$ denotam, respectivamente, o centro e raio de cada bola, estamos interessados em resolver o seguinte problema de otimização

$$\underset{x \in \mathbb{R}^n}{\text{minimizar}}\, f(i_1, \ldots, i_m; x) := \|x^{i_m} - x^{i_1}\| + \sum_{\nu=1}^{m-1} \|x^{i_\nu} - x^{i_{\nu+1}}\| \quad \text{sujeito a} \quad x^{i_\nu} \in \mathcal{B}_{i_\nu} \text{ para } \nu = 1, \ldots, m, \tag{3.1}$$

em que $\{i_1, i_2, \ldots, i_m\}$ é uma permutação de $\{1, 2, \ldots, m\}$, $n = 2m$, $x^{i_\nu} = (x_1^{i_\nu}, x_2^{i_\nu}) \in \mathbb{R}^2$ para $\nu = 1, \ldots, m$, $x^T = ((x^1)^T, \ldots, (x^m)^T) \in \mathbb{R}^n$ e $\|\cdot\|$ corresponde à norma Euclidiana.

A resolução de (3.1) pode ser dividida em duas partes:

(i) Determinação de uma permutação, que corresponde à um problema discreto. Isto é, fixado um conjunto de pontos $x^1, \ldots, x^m$, (3.1) corresponde ao TSP Euclidiano em $\mathbb{R}^2$. Portanto, podemos empregar neste caso heurísticas já desenvolvidas para o TSP discreto, como àquelas discutidas no Capítulo 2, para obter uma permutação $i_1{}^*, i_2{}^*, \ldots, i_m{}^*$ que minimiza o custo $f(i_1{}^*, \ldots, i_m{}^*; x)$ para cada x fixado;

(ii) Obtenção dos pontos a serem visitados, que consiste num problema de otimização contínua. Ou seja, fixada uma permutação $i_1, i_2, \ldots, i_m$, o problema (3.1) se torna um problema de otimização contínua. Para este caso, propomos, baseados em [6], utilizar métodos de busca em bloco de coordenadas para encontrar uma solução x^*.

Assim, nosso objetivo é combinar as estratégias de (i) e (ii) na construção de algoritmos para o CTSP (3.1).

É interessante salientar que o CTSP, como considerado em [6], é uma generalização de outro problema presente na literatura: o problema do caixeiro viajante *suficientemente próximo* (CETSP, do inglês *close-enough traveling salesman problem* [11, 20, 22]). No CETSP, o tour não precisa passar exatamente por cada ponto e sim dentro de uma distância pré-especificada de cada um deles, ou seja, *suficientemente próximo*. Formalmente, sejam $p_0, c_1, \ldots, c_m \in \mathbb{R}^2$ pontos distintos e $\Omega_i \in \mathbb{R}^2$ um conjunto compacto que contém c_i, $i = 1, \ldots, m$. O CETSP consiste em determinar os pontos $x_i \in \Omega_i$, $i = 1, \ldots, m$, e uma sequência $\mathcal{S} = \{i_1, i_2, \ldots, i_m\}$ de modo que o tour iniciando em p_0 (chamado de *depósito*), passando por cada um dos pontos x_i na ordem definida pela sequência $\mathcal{S}$ e retornando para p_0 tenha comprimento mínimo (em relação à distância Euclidiana).

Note que, no CETSP os conjuntos Ω_i são definidos por uma curva de nível de uma função de distância, enquanto que no CTSP tais conjuntos são arbitrários, podendo assumir a forma de polígonos quaisquer por exemplo. Além disso, o CETSP se resume à versão clássica do TSP Euclidiano quando cada conjunto Ω_i é constituído de um único ponto e, portanto, ele também pertence à classe dos problemas NP-difícil.

A formulação dada pelo CETSP apresenta uma ampla aplicabilidade, pois pode ser utilizada para modelar vários problemas do mundo real. Um exemplo é no uso de tecnologias de identificação por radiofrequência para leitura automatizada de medidores. Shuttleworth et al. [31] e Dong et al. [11] abordam esta aplicação utilizando o CETSP e em [11] ele é formulado como um problema de programação não linear inteira mista (PNLIM). Dentro de um âmbito mais teórico, Behdani and Smith [4] também formulam o CETSP como um problema de PNLIM e propõem métodos para gerar limites inferior e superior para o custo de um tour ótimo do CETSP. Outros exemplos de uso do CETSP que aparecem na literatura são na localização de incêndios florestais por veículos aéreos não tripulados [26] e no reconhecimento de diagnóstico de painéis solares [25].

Na próxima seção, discutiremos a teoria dos métodos de busca em coordenadas, pois um método desta classe será utilizado nos algoritmos propostos para resolver o problema (3.1).

3.2 Métodos de busca em coordenadas

Seja $x = (x_1, \ldots, x_n)$ um vetor do $\mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$ uma função contínua e considere o seguinte problema de otimização irrestrita

$$\underset{x \in \mathbb{R}^n}{\text{minimizar}}\, f(x). \tag{3.2}$$

Métodos de busca em coordenadas (CDM, do inglês *coordinate descent methods* [5, 19, 24]) para resolver (3.2) são métodos iterativos em que cada iterando x^k é a aproximação de um minimizador do problema

$$\underset{x_i}{\text{minimizar}}\, f(x_1, x_2, \ldots, x_n), \tag{3.3}$$

para algum $i \in \{1, \ldots, n\}$. Ou seja, a cada iteração k somente uma componente x_i do vetor x^k será alterada para minimizar a função objetivo. A sequência dos iterandos gerados por tais métodos será da forma

$$x^{k+1} = x^k + t_k e_{i_k}, \tag{3.4}$$

em que $t_k \in \mathbb{R}$ é o tamanho de passo, i_k é o índice da variável alterada durante a k-ésima iteração e e_i corresponde ao i-ésimo vetor da base canônica de $\mathbb{R}^n$. Quando mais de uma coordenada é alterada a cada iteração, falamos em métodos de busca em bloco de coordenadas (BCDM, do inglês *block coordinate descent methods*). Neste caso, a notação utilizada em (3.4) pode ser estendida

considerando uma partição $x = (x_1, \ldots, x_m)$ do vetor de variáveis x e da matriz identidade $I_n = (U_1\ U_2 \ldots U_m)$, com $U_i \in \mathbb{R}^{n \times n_i}$ para $i = 1, \ldots, m$, em que m corresponde ao número de blocos, $x_i \in \mathbb{R}^{n_i}$ é o i-ésimo bloco, n_i é o tamanho do i-ésimo bloco e $n_1 + n_2 + \cdots + n_m = n$. Assim, podemos reescrever (3.4) da forma

$$x^{k+1} = x^k + U_{i_k} u^k,$$

em que u^k corresponde ao vetor de passos.

De modo geral, os métodos de busca em coordenadas são compostos de duas etapas principais: (i) a escolha da coordenada ou bloco de coordenadas e (ii) a atualização do bloco escolhido. As diferentes estratégias que podem ser empregadas nestas etapas resultando em diferentes métodos. Na escolha *cíclica* de coordenadas basta que uma dada sequência de índices sempre se repita. Um exemplo é a escolha trivial dada pela sequência $\{1, 2, \ldots, n, 1, \ldots\}$, isto é, os termos da sequência de índices das coordenadas são [30]

$$\begin{aligned} i_0 &= 1, \\ i_k &= \begin{cases} 1, & \text{se} \quad i_{k-1} = n \\ i_{k-1} + 1, & \text{caso contrário.} \end{cases} \end{aligned} \tag{3.5}$$

A vantagem dessa abordagem está no fato de não depender da avaliação do gradiente $\nabla f(x^k)$ da função objetivo para determinar a direção de descida a cada iteração.

Outra abordagem é dada pelo método de *Gauss-Southwell*. Neste método a coordenada escolhida será aquela que corresponde à componente do vetor gradiente de maior valor absoluto. Formalmente,

$$i_k = \arg\max_{1 \leq i \leq n} |\nabla_{x_i^k} f(x^k)|,$$

em que $\nabla_{x_i^k} f(x^k)$ corresponde à i-ésima componente do vetor gradiente $\nabla f(x^k)$.

A escolha das coordenadas também pode ser feita de forma aleatória. Este é o caso do método proposto no trabalho de Nesterov [23] para o problema (3.2) com f convexa e diferenciável. Em seu algoritmo de escolha aleatória, Nesterov [23] assume que as derivadas parciais da função objetivo são *Lipschitz* contínuas com constantes L_i e a escolha da coordenada a ser atualizada na k-ésima iteração é realizada com uma probabilidade p dada pela fórmula

$$p(i_k = i) = L_i^\alpha . \left[\sum_{j=1}^{n} L_j^\alpha \right]^{-1}, \quad i = 1, \ldots, n, \tag{3.6}$$

em que $\alpha \in [0, 1]$ é um parâmetro que controla a relevância das constantes L_i para a escolha dos blocos. Observe que se $\alpha = 0$ o algoritmo de escolha tem distribuição uniforme.

Em relação a atualização do bloco de coordenadas, uma estratégia é a escolha ótima

$$x_i^k = \arg\min_{\xi} f(x_1^k, \ldots, x_{i-1}^k, \xi, x_{i+1}^k, \ldots, x_n^k)$$

ou, reescrevendo em função do vetor de passos u_k, assim como em [30],

$$u_k = \arg\min_{u} f(x^k + U_{i_k} u). \tag{3.7}$$

No entanto, esta abordagem só é viável no caso em que resolver (3.7) é uma tarefa fácil. Em contrapartida, oferece a vantagem de que não requer a diferenciabilidade da função objetivo f.

Outra estratégia consiste em dar um passo de tamanho fixo na direção oposta a do gradiente do bloco da seguinte forma

$$u^k = -\frac{1}{L} \nabla_{x_i^k} f(x^k), \tag{3.8}$$

em que L_i é a constante de Lipschitz associada às derivadas parciais do i-ésimo bloco. Esta é a estratégia empregada nos trabalhos de Nesterov [23] e Beck and Tetruashvili [3]. Em [23], o passo (3.8) é aplicado com escolha aleatória do bloco de coordenadas (3.6) e resultados teóricos de convergência são apresentados para os casos em que a função objetivo em (3.2) é convexa e diferenciável ou fortemente convexa. Já Beck and Tetruashvili [3] analisam a convergência do método de busca em coordenadas que utiliza (3.8) e a escolha cíclica (3.5).

No caso em que f é uma função quadrática e convexa, Santos [30] propõe obter o vetor de passos u^k resolvendo o sistema

$$\nabla^2_{x_i^k} f(x^k) u^k = -\nabla_{x_i^k} f(x^k),$$

o que exigiria o cálculo da matriz Hessiana $\nabla^2_{x_i^k} f(x^k)$, do gradiente $\nabla_{x_i^k} f(x^k)$ e $\mathcal{O}(n_{i_k}^3)$ operações para resolver o sistema acima. No entanto, para calcular u_k no caso em que $n_{i_k} = 1$ seria preciso computar apenas uma entrada da matriz Hessiana e uma componente do vetor gradiente.

A vantagem dos métodos de busca em coordenadas e a motivação por trás da sua escolha para resolver o TSP contínuo está na simplicidade de suas iterações, pois os subproblemas (3.3) gerados a cada iteração tem dimensão menor que (3.2) e podem ser mais baratos de resolver. Este é o caso em problemas com grande número de variáveis e nos quais a avaliação do gradiente $\nabla f(x^k)$ e da função objetivo $f(x^k)$ a cada iteração se configuram em operações muito custosas ou até inviáveis. Algumas aplicações relevantes e resultados teóricos de convergência podem ser encontrados nos trabalhos de Wright [32] e Santos [30].

Algumas estratégias dos métodos de busca em bloco de coordenadas apresentadas até aqui

também podem ser estendidas aos problemas de otimização com restrições. Considere o problema

$$\text{minimizar } f(x) \quad \text{sujeito a} \quad x \in \Omega. \tag{3.9}$$

Suponha que o conjunto viável Ω pode ser escrito como o produto cartesiano de conjuntos convexos e fechados $\Omega_1, \ldots, \Omega_m$, em que cada $\Omega_i \in \mathbb{R}^{n_i}$ e $n = n_1 + \cdots + n_m$. Considere também uma partição do vetor x da forma $x = (x_1, x_2, \ldots, x_m)$, com $x_i \in \mathbb{R}^{n_i}$ para $i = 1, \ldots, m$. Diante disso, a restrição $x \in \Omega$ é equivalente a $x_i \in \Omega_i$ para $i = 1, \ldots, m$.

O método de *Gauss-Sidel* não-linear apresentado em [5] é um exemplo de método de busca em bloco de coordenadas que pode ser aplicado a problemas com restrições como (3.9). Neste método, cada iterando x^{k+1} é obtido a partir do anterior resolvendo

$$x_i^{k+1} = \arg\min_{\xi \in \Omega_i} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, \xi, x_{i+1}^{k}, \ldots, x_m^{k}), \quad i = 1, \ldots, m,$$

isto é, a cada iteração o custo é minimizado com respeito a cada um dos blocos de coordenadas x_i^k tomados em ordem cíclica. A diferença neste caso é que todos os blocos de coordenadas são atualizados para completar uma iteração. Um resultado de convergência deste método, para problemas como o especificado acima, é fornecido por Bertsekas [5], sob a hipótese de que o minimizador ao longo de qualquer direção, respectiva aos blocos de coordenadas, é único. A necessidade desta hipótese fica clara através de um exemplo em $\mathbb{R}^3$ fornecido por Powell [27, p. 195]. Com este exemplo, Powell [27] demonstra que sem as devidas hipóteses o método acima com escolha cíclica de coordenadas pode não convergir a um ponto estacionário.

Observe que o CTSP tem o formato de (3.9), basta tomar $\Omega_i = \mathcal{B}_i$ e $n_i = 2$ para $i = 1, \ldots, m$. Portanto, fixada uma permutação, podemos empregar as estratégias discutidas nesta seção para resolvê-lo numericamente. Na Seção 3.3 a seguir, apresentamos em detalhes os métodos e estratégias pensados para o CTSP (3.1).

3.3 Algoritmos para o TSP contínuo

Nesta seção vamos elaborar as estratégias e algoritmos para o problema do caixeiro viajante contínuo. Como abordado na Seção 3.1, propomos resolvê-lo numericamente através de um esquema iterativo no qual a determinação de cada iterando $(i_1^k, \ldots, i_m^k, x^k)$ ocorre em duas etapas. Em resumo, o algoritmo inicia com um conjunto inicial de m pontos $x_i \in \mathcal{B}_i$ para $i = 1, \ldots, m$. Com estes pontos fixados, uma permutação inicial $T^0 := (i_1^0, \ldots, i_m^0)$ é construída através de uma heurística construtiva. Em seguida, os pontos x_i são recalculados com o método de busca em coordenadas considerando a permutação dada por T^0. Tais pontos irão compor o vetor $x^0 := (x_1^0, \ldots, x_m^0)$.

Com isso obtemos o tour inicial (T^0, x^0). A partir disso, o algoritmo segue iterativamente de modo que a cada iteração desejamos encontrar um par (T^k, x^k) com custo menor do que o da iteração anterior. Essa busca se faz testando permutações (obtidas com métodos de busca local) e, para cada permutação, calculando um novo x com o método de busca em coordenadas para determinar o comprimento do tour dessa permutação. Vale ressaltar que a palavra "tour" será usada para se referir ao caminho gerado ao se percorrer os pontos de $x = (x_1, \ldots, x_m)$ na ordem dada pelo vetor T, e que o comprimento total de um tour (T, x) é dado pelo custo da função objetivo f em (3.1), isto é,

$$f(T, x) = \left(\sum_{\nu=1}^{m-1} \|x_{T_\nu} - x_{T_{\nu+1}}\| \right) + \|x_{T_m} - x_{T_1}\|,$$

em que m corresponde ao número de bolas do problema e $\|\cdot\|$ é a norma Euclidiana.

Vejamos agora os detalhes dos algoritmos empregados em cada etapa do processo descrito acima, iniciando pelo método de busca em coordenadas.

Problema contínuo

Considere inicialmente o CTSP supondo que uma dada permutação $T = (i_1, i_2, \ldots, i_m)$ está fixada. Neste caso, o problema (3.1) consiste em determinar os pontos $x_i \in \mathcal{B}_i$, $i = 1, \ldots, m$, de modo que o tour gerado ao visitá-los na ordem dada por T fixada tenha comprimento mínimo. Observe que cada ponto $x^i = (x_1^i, x_2^i)$ pertence ao $\mathbb{R}^2$ e, portanto, a dimensão do problema é dada pelo dobro do número de bolas. Vale ressaltar que no CTSP não há necessidade de que os conjuntos sejam disjuntos. No entanto, como desejamos a diferenciabilidade da função objetivo, vamos supor neste trabalho que as bolas $\mathcal{B}_i$, que formam o conjunto viável, são disjuntas. Dessa forma, temos que a função objetivo $f(T, x) = f(x)$ e $f : \mathbb{R}^{2m} \to \mathbb{R}$ é contínua e diferenciável e seu gradiente $\nabla f(x)$ é um vetor pertencente ao $\mathbb{R}^{2m}$ dado por

$$\nabla f(x) = \begin{bmatrix} \dfrac{(x_1^{i_1} - x_1^{i_2})}{\sqrt{(x_1^{i_1} - x_1^{i_2})^2 + (x_2^{i_1} - x_2^{i_2})^2}} + \dfrac{(x_1^{i_1} - x_1^{i_m})}{\sqrt{(x_1^{i_m} - x_1^{i_1})^2 + (x_2^{i_m} - x_2^{i_1})^2}} \\ \dfrac{(x_2^{i_1} - x_2^{i_2})}{\sqrt{(x_1^{i_1} - x_1^{i_2})^2 + (x_2^{i_1} - x_2^{i_2})^2}} + \dfrac{(x_2^{i_1} - x_2^{i_m})}{\sqrt{(x_1^{i_m} - x_1^{i_1})^2 + (x_2^{i_m} - x_2^{i_1})^2}} \\ \vdots \\ \dfrac{(x_1^{i_m} - x_1^{i_{m-1}})}{\sqrt{(x_1^{i_{m-1}} - x_1^{i_m})^2 + (x_2^{i_{m-1}} - x_2^{i_m})^2}} + \dfrac{(x_1^{i_m} - x_1^{i_1})}{\sqrt{(x_1^{i_m} - x_1^{i_1})^2 + (x_2^{i_m} - x_2^{i_1})^2}} \\ \dfrac{(x_2^{i_m} - x_2^{i_{m-1}})}{\sqrt{(x_1^{i_{m-1}} - x_1^{i_m})^2 + (x_2^{i_{m-1}} - x_2^{i_m})^2}} + \dfrac{(x_2^{i_m} - x_2^{i_1})}{\sqrt{(x_1^{i_m} - x_1^{i_1})^2 + (x_2^{i_m} - x_2^{i_1})^2}} \end{bmatrix},$$

cujas entradas correspondem às derivadas parciais de f em relação a cada uma das componentes do vetor $x^T = ((x^{i_1})^T, \ldots, (x^{i_m})^T) = (x_1^{i_1}, x_2^{i_1}, x_1^{i_2}, \ldots, x_1^{i_m}, x_2^{i_m})$. Note que, o cálculo de cada entrada

do gradiente acima exige cerca de 17 operações e a avaliação de duas raízes quadradas. Logo, seu cálculo pode se configurar numa tarefa custosa conforme o número de bolas aumenta, sobretudo nos métodos em que o gradiente da função objetivo é reavaliado a cada iteração. Ademais, a função objetivo de (3.1) é tal que reavaliá-la num ponto x que teve apenas algumas componentes x^i modificadas é uma tarefa mais simples do sua avaliação integral. Estas são algumas características que motivaram nossa escolha pelo método de busca em bloco de coordenadas para resolver esse problema. Vejamos então como podemos reescrever o problema para aplicar estratégias deste método.

Seguindo a ideia dos métodos de busca em bloco de coordenadas (Seção 3.2), propomos minimizar a função objetivo f de (3.1) em relação a um bloco de coordenadas (ponto) x^i por vez, tal que a escolha do bloco é feita de acordo com a ordem estabelecida pela permutação T que está fixada. Ou seja, para cada $\nu = 1, \ldots, m$, desejamos determinar o ponto $x^{i_\nu} \in \mathcal{B}_{i_\nu}$ que minimiza a distância até os pontos fixados $x^{i_{\nu-1}}$ e $x^{i_{\nu+1}}$.

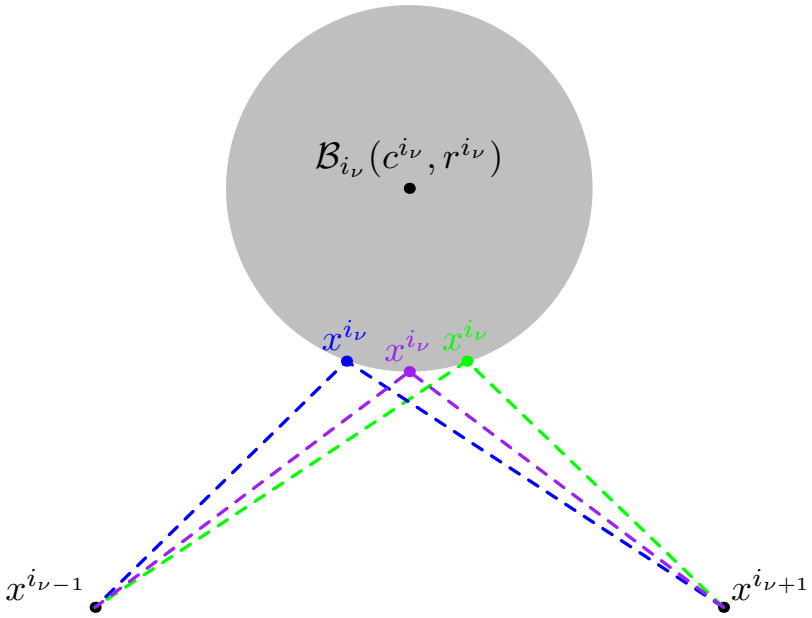


Figura 3.1: Exemplo ilustrativo de pontos viáveis x^{i_ν}.

Matematicamente, o problema pode ser reformulado da forma

$$\underset{x^{i_\nu} \in \mathbb{R}^2}{\text{minimizar}} \, f(x) \quad \text{sujeito a} \quad x^{i_\nu} \in \mathcal{B}_{i_\nu}$$

ou, equivalentemente,

$$\underset{x^{i_\nu} \in \mathbb{R}^2}{\text{minimizar}} \, \varphi(x^{i_\nu}) := \|a - x^{i_\nu}\| + \|x^{i_\nu} - b\| \quad \text{sujeito a} \quad x^{i_\nu} \in \mathcal{B}_{i_\nu}, \tag{3.10}$$

para cada $\nu = 1, \ldots, m$, em que a e $b \in \mathbb{R}^2$ correspondem aos pontos anterior e posterior a x^{i_ν} de

acordo com a permutação T. Vamos assumir que $x^{i_{\nu-1}} = x^{i_n}$ para $\nu = 1$ e $x^{i_{\nu+1}} = x^{i_1}$ para $\nu = n$.

Definindo $a = (a_1, a_2)$, $b = (b_1, b_2)$ e denotando por $c^{i_\nu} = (c_1^{i_\nu}, c_2^{i_\nu})$ e r^{i_ν} o centro e raio associados à bola $\mathcal{B}_{i_\nu}$, podemos reescrever (3.10) da forma

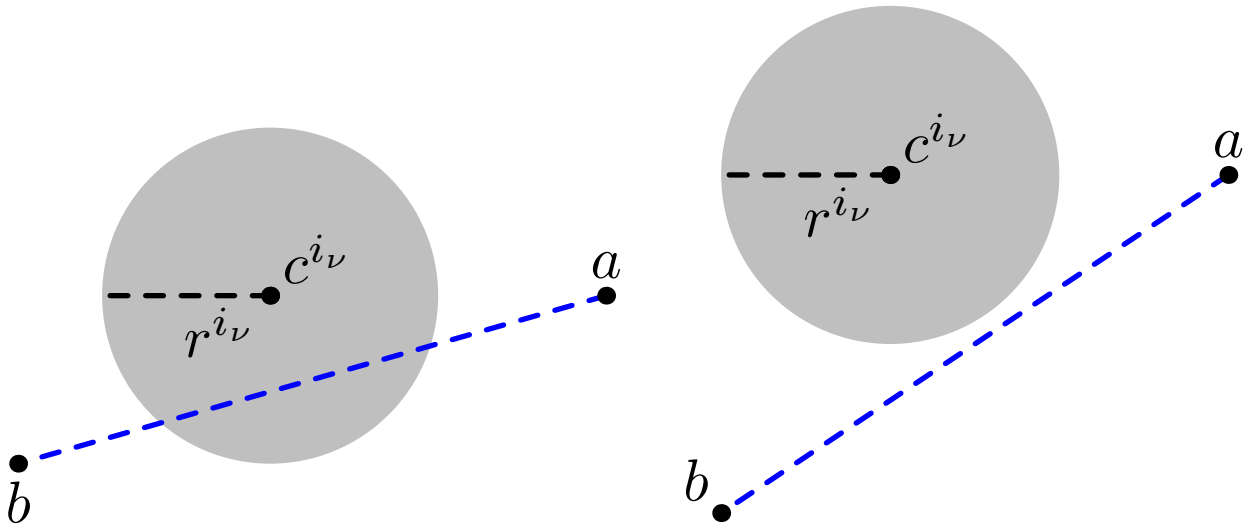
$$\begin{aligned} \underset{x^{i_\nu} \in \mathbb{R}^2}{\text{minimizar}} \quad & \varphi(x^{i_\nu}) := \sqrt{(a_1 - x_1^{i_\nu})^2 + (a_2 - x_2^{i_\nu})^2} + \sqrt{(b_1 - x_1^{i_\nu})^2 + (b_2 - x_2^{i_\nu})^2} \\ \text{sujeito a} \quad & (c_1^{i_\nu} - x_1^{i_\nu})^2 + (c_2^{i_\nu} - x_2^{i_\nu})^2 - (r^{i_\nu})^2 \leq 0. \end{aligned} \tag{3.11}$$

O gradiente de $\varphi(x^{i_\nu}) : \mathbb{R}^2 \to \mathbb{R}^2$ é dado por

$$\nabla\varphi(x^{i_\nu}) = \begin{bmatrix} \dfrac{\partial\varphi}{\partial x_1^{i_\nu}} \\ \dfrac{\partial\varphi}{\partial x_2^{i_\nu}} \end{bmatrix} = \begin{bmatrix} \dfrac{(x_1^{i_\nu} - a_1)}{\sqrt{(a_1 - x_1^{i_\nu})^2 + (a_2 - x_2^{i_\nu})^2}} + \dfrac{(x_1^{i_\nu} - b_1)}{\sqrt{(b_1 - x_1^{i_\nu})^2 + (b_2 - x_2^{i_\nu})^2}} \\ \dfrac{(x_2^{i_\nu} - a_2)}{\sqrt{(a_1 - x_1^{i_\nu})^2 + (a_2 - x_2^{i_\nu})^2}} + \dfrac{(x_2^{i_\nu} - b_2)}{\sqrt{(b_1 - x_1^{i_\nu})^2 + (b_2 - x_2^{i_\nu})^2}} \end{bmatrix}. \tag{3.12}$$

Como as bolas são disjuntas o gradiente acima é não nulo e, portanto, está bem definido.

Inicialmente, observe que o problema (3.11) admite soluções globais, pois satisfaz as hipóteses do Teorema de *Weiestrass*: a função objetivo φ é contínua e o conjunto viável $\mathcal{B}_{i_\nu}$ compacto. No entanto, como a Figura 3.2 a seguir ilustra, duas situações podem ocorrer: o segmento de reta $[a, b]$ entre os pontos $a = (a_1, a_2)$ e $b = (b_1, b_2)$ intersecta a bola $\mathcal{B}_{i_\nu}$ (Figura 3.2a), ou não (Figura 3.2b).



(a) Infinitas soluções.

(b) A solução é única.

Figura 3.2: A figura da esquerda ilustra um exemplo no qual o problema (3.11) admite infinitas soluções, enquanto que na figura da direita temos um exemplo em que a solução é única.

No caso em que o segmento $[a, b]$ intersecta a bola $\mathcal{B}_{i_\nu}$ como na Figura 3.2a sabemos, pela desigualdade triangular, que os pontos que minimizam a distância até os extremos a e b são os pontos que pertencem ao segmento e à bola. Assim, quando esta configuração ocorre, o problema (3.11) admite infinitas soluções $x^{i_\nu *}$, todas pertencentes ao segmento $[a, b]$ e para cada uma existe

$\lambda^* \in (0,1)$ tal que ${x^{i_\nu}}^* = \lambda^* a + (1-\lambda^*)b$. Como neste caso as soluções são infinitas, para manter uma certa uniformidade na escolha dada pelo algoritmo, vamos escolher a solução dada pela projeção ortogonal do centro c^{i_ν} sobre o segmento $[a, b]$. Este processo é descrito no Algoritmo 1 a seguir. Em nossos algoritmos o critério de parada considerará a distância entre iterandos consecutivos do método e, portanto, optamos por essa escolha determinística para que esse critério tenha chances de se satisfazer.

Algoritmo 1: Calcula a projeção ortogonal do centro c de uma bola do $\mathbb{R}^2$ sobre o segmento de reta entre pontos a e $b \in \mathbb{R}^2$.

Entrada: $a = (a_1, a_2)$, $b = (b_1, b_2)$, $c = (c_1, c_2)$.
Saída: $c^P = (c_1^P, c_2^P)$.

Função ProjecaoEmAB(a, b, c, c^P)
  $t \leftarrow \dfrac{(c-a)^T(b-a)}{\|b-a\|_2^2}$ e $\hat{c} \leftarrow a + t(b-a)$
  se $|a_2 - b_2| < |a_1 - b_1|$ **então**
    $\lambda \leftarrow \dfrac{\hat{c}_1 - b_1}{a_1 - b_1}$
  senão
    $\lambda \leftarrow \dfrac{\hat{c}_2 - b_2}{a_2 - b_2}$
  $\lambda^* := \max(0, \min(\lambda, 1))$ e $c^P := \lambda^* a + (1-\lambda^*)b$
  retorne c^P

Por conseguinte, temos o caso da Figura 3.2b, no qual a bola $\mathcal{B}_{i_\nu}$ e o segmento de reta $[a, b]$ possuem intersecção vazia. Afirmamos que nesta situação a solução ${x^{i_\nu}}^*$ de (3.11) pertencerá à fronteira de $\mathcal{B}_{i_\nu}$. Com efeito, tome $z = (z_1, z_2) \in \operatorname{int} \mathcal{B}_{i_\nu}$ arbitrário. Do fato de z estar no interior da bola, toda direção $d \in \mathbb{R}^2$ será viável a partir de z. Ademais, como visto anteriormente, o $\nabla\varphi(z)$ dado pela equação (3.12) é diferente de zero, pois as bolas são disjuntas. Tomando a direção do menos gradiente $d = -\nabla\varphi(z)$, obtemos uma direção viável e de descida a partir de z, pois

$$-\nabla\varphi(z)^T \nabla\varphi(z) = -\|\nabla\varphi(z)\|^2 < 0.$$

Logo, z não pode ser solução de (3.11) e concluímos que a solução do nosso problema deverá pertencer à fronteira da bola $\mathcal{B}_{i_\nu}$.

Portanto, neste contexto a solução ${\boldsymbol{x}^{i_\nu}}^*$ estará sobre o bordo de $\mathcal{B}_{i_\nu}$ e a restrição de desigualdade no problema (3.11) será ativa nesse ponto. A partir disso, podemos limitar o conjunto viável aos pontos $\boldsymbol{x}^{i_\nu}$ da fronteira $\partial\mathcal{B}_{i_\nu}$ e parametrizá-los em função do centro $\boldsymbol{c}^{i_\nu} = (c_1^{i_\nu}, c_2^{i_\nu})$ e do raio r^{i_ν} de $\mathcal{B}_{i_\nu}$ através da seguinte fórmula

$$\boldsymbol{x}^{i_\nu} = \begin{bmatrix} \boldsymbol{x}_1^{i_\nu} \\ \boldsymbol{x}_2^{i_\nu} \end{bmatrix} = \begin{bmatrix} c_1^{i_\nu} + r^{i_\nu} \cos\theta \\ c_2^{i_\nu} + r^{i_\nu} \sin\theta \end{bmatrix}. \tag{3.13}$$

Substituindo (3.13) em (3.11), convertemos (3.11) em um problema de minimização unidimensional equivalente mas sem restrições, dado por

$$\underset{\theta\in[0,2\pi]}{\text{minimizar}} \sqrt{(a_1 - c_1^{i_\nu} - r^{i_\nu}\cos\theta)^2 + (a_2 - c_2^{i_\nu} - \sin\theta)^2} + \sqrt{(b_1 - c_1^{i_\nu} - r^{i_\nu}\cos\theta)^2 + (b_2 - c_2^{i_\nu} - r^{i_\nu}\sin\theta)^2}. \tag{3.14}$$

Como (3.14) corresponde a um problema de otimização sem restrições, com função objetivo contínua e bem definida, sabemos pelas condições necessárias de otimalidade (facilmente encontradas em livros clássicos de otimização como [5, 19, 24]), que se θ^* é um minimizador de (3.14), então $\nabla\varphi(\theta^*) = 0$. Métodos numéricos para obter soluções neste caso atuam procurando pontos estacionários do problema, isto é, pontos cujo gradiente é nulo. Tendo isso em vista, propomos então utilizar o método de Newton para resolver (3.14). Um pseudocódigo para o método de Newton globalizado [21] é apresentado no Algoritmo 2.

Algoritmo 2: Newton globalizado com busca linear para minimização de funções unidimensionais.

Entrada: $\alpha \in (0,1)$, $\beta > 0$, $0 < \sigma_1 \leq \sigma_2 < 1$, $\epsilon > 0$, $\ell_{\max} \geq 0$, $\theta_0 \in \mathbb{R}$.
Saída: θ^*.

Função Newton(α, β, σ_1, σ_2, ϵ, $\ell_{\max}$, θ_0, θ^*)
  $\ell \leftarrow 0$
  enquanto $|\varphi'(\theta_\ell)| > \epsilon$ e $\ell < \ell_{\max}$ **faça**
    se $\varphi''(\theta_\ell) \leq 0$ **então**
      $d_\ell := -\varphi'(\theta_\ell)$
    senão
      $d_\ell := -\varphi'(\theta_\ell)/\varphi''(\theta_\ell)$
    se $|d_\ell| < \beta\,|\varphi'(\theta_\ell)|$ **então**
      $d_\ell := \beta\,|\varphi'(\theta_\ell)|\,(d_\ell/|d_\ell|)$
    $t \leftarrow 1$
    enquanto $\varphi(\theta_\ell + t d_\ell) \nleq \varphi(\theta_\ell) + \alpha t \varphi'(\theta_\ell) d_\ell$ **faça**
      $t \leftarrow \dfrac{-t^2\varphi'(\theta_\ell)d_\ell}{2[\varphi(\theta + td_\ell) - \varphi(\theta_\ell) - t\varphi'(\theta_\ell)d_\ell]}$
      se $t \notin [\sigma_1\,t, \sigma_2\,t]$ **então**
        $t \leftarrow t/2$
    $\theta_{\ell+1} := \theta_\ell + t\,d_\ell$
    $\ell \leftarrow \ell + 1$
  $\theta^* := \theta_\ell$
  retorne θ^*

Por fim, é preciso desenvolver uma estratégia para decidir se o problema (3.11) deve ser resolvido utilizando o Algoritmo 1 ou o Algoritmo 2. Para tanto, propomos avaliar se o segmento $[a, b]$ intersecta a bola $\mathcal{B}_{i_\nu}$ ou não. Assim, se a projeção c^P do centro c^{i_ν} em $[a, b]$ satisfaz $\|c^P - c^{i_\nu}\| \leq r^{i_\nu}$ temos que $c^P \in \mathcal{B}_{i_\nu}$ e o Algoritmo 1 será aplicado para determinar uma solução de (3.11), que será justamente o ponto c^P. Caso contrário, o Algoritmo 2 de Newton será utilizado.

Portanto, nosso método de busca em bloco de coordenadas para o CTSP dado pelo Algoritmo 3 consiste em empregar as estratégias acima para resolver o subproblema (3.11) para cada x^{i_ν} na ordem cíclica dada por $\nu = \{1, \dots, m, 1, \dots\}$ até que um critério de parada seja satisfeito. A cada vez que todos os pontos x^{i_ν} são atualizados, dizemos que um ciclo do método foi realizado.

Algoritmo 3: Método de busca em coordenadas para otimizar os pontos de uma rota.

Entrada: $\alpha \in (0,1)$, $\beta > 0$, $0 < \sigma_1 \leq \sigma_2 < 1$, $\epsilon^{\mathrm{N}} > 0$, $\ell^{\mathrm{N}}_{\max} \geq 0$, m, $c = (c_1, \dots, c_m)$, $r = (r_1, \dots, r_m)$, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, $\theta = (\theta_1, \dots, \theta_m)$, T, $x^0 = (x^0_1, \dots, x^0_m)$.

Saída: $x^* = (x^*_1, \dots, x^*_m)$.

Função BuscaEmCoordenadas(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, m, c, r, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, θ, T, x^0, x^*)
- $\ell \leftarrow 0$
- **faça**
  - **para** $k \leftarrow 1$ **até** m **faça**
    - **se** $k = 1$ **então**
      - $a \leftarrow x^{\ell}_{T_m}$
      - $b \leftarrow x^{\ell}_{T_{k+1}}$
    - **senão se** $k = m$ **então**
      - $a \leftarrow x^{\ell+1}_{T_{k-1}}$
      - $b \leftarrow x^{\ell}_{T_1}$
    - **senão**
      - $a \leftarrow x^{\ell+1}_{T_{k-1}}$
      - $b \leftarrow x^{\ell}_{T_{k+1}}$
    - $r \leftarrow r_{T_k}$ e $c \leftarrow c_{T_k}$
    - ProjecaoEmAB(a, b, c, c^P)
    - **se** $\|c - c^P\|_2 \leq r$ **então**
      - $x^{\ell+1}_{T_k} \leftarrow c^P$
    - **senão**
      - $\theta_0 \leftarrow \theta_{T_k}$
      - Newton(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, θ_0, θ^*)
      - $\theta_{T_k} \leftarrow \theta^*$
      - $x^{\ell+1}_{T_k} \leftarrow c + r \begin{pmatrix} \cos\theta^* \\ \sin\theta^* \end{pmatrix}$
    - $\delta \leftarrow |f(T, x^{\ell}) - f(T, x^{\ell-1})|$
    - $\ell \leftarrow \ell + 1$
  - **enquanto** $\delta > \epsilon^{\mathrm{BC}}$ e $\ell < \ell^{\mathrm{BC}}_{\max}$
  - $x^* := x^{\ell}$
  - **retorne** x^*

Vejamos agora os algoritmos iterativos que, empregando heurísticas para obter as permutações T^k e o método acima para calcular os pontos x^k, pretendem gerar soluções (T^*, x^*) para o problema (3.1).

Problema discreto

Vamos considerar agora os algoritmos para se obter uma solução (T^*, x^*) para o problema (3.1). Para tanto, empregaremos heurísticas do TSP discreto apresentadas no Capítulo 2 junto do método BCD (3) descrito na seção anterior. Como comentado anteriormente, as heurísticas de busca local são heurísticas de melhoria, que usam um tour inicial e através de uma busca em vizinhanças do tour atual procuram obter tours melhores, isto é, de menor comprimento. Portanto, iniciamos com o algoritmo da heurística construtiva para gerar o tour inicial.

Para construir o tour inicial T^0 vamos empregar a heurística do vizinho mais próximo (NN) descrita na Seção 2.1.1. O Algoritmo 4 apresenta o pseudocódigo dessa heurística adaptada ao contexto do problema (3.1). Nele temos que m corresponde ao número de bolas a serem visitadas, $D = (d_{ij})$ é a matriz cujas entradas correspondem às distâncias entre os centros das bolas e (T^0, x^0) será o tour inicial gerado ao final do processo. Inicialmente, o algoritmo constrói T^0 pelo método NN considerando as distâncias entre os centros das bolas para decidir a ordem com a qual elas serão visitadas. Em seguida, aplica o método de busca em bloco de coordenadas dado pelo Algoritmo 3 para otimizar os pontos visitados em cada bola e definir x^0.

Algoritmo 4: Heurística construtiva do vizinho mais próximo e o método de busca em coordenadas para construir o tour inicial (T^0, x^0).

Entrada: $\alpha \in (0,1)$, $\beta > 0$, $0 < \sigma_1 \leq \sigma_2 < 1$, $\epsilon^{\mathrm{N}} > 0$, $\ell^{\mathrm{N}}_{\max} \geq 0$, m, $c = (c_1, \ldots, c_m)$, $r = (r_1, \ldots, r_m)$, $\epsilon^{\mathrm{BC}} > 0$, $\ell^{\mathrm{BC}}_{\max} \geq 0$, $\theta = (\theta_1, \ldots, \theta_m)$, $D = (d_{ij})_{m \times m}$.
Saída: T^0, $x^0 = (x^0_1, \ldots, x^0_m)$.

Função VizinhoMaisProximo(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, m, c, r, θ, T^0, x^0)
  Escolha $\delta \in \{1, \ldots, m\}$
  $T^0_1 := \delta$ e $U_\delta \leftarrow 0$
  $U_k \leftarrow k$ para $k \in \{1, \ldots, \delta - 1, \delta + 1, \ldots, m\}$
  para $i \leftarrow 1$ **até** $m - 1$ **faça**
    $d_{\mathrm{best}} \leftarrow \|D\|_\infty$
    para $k \leftarrow 1$ **até** m **faça**
      se $U_k \neq 0$ **então**
        se $d_{T_i \times k} < d_{\mathrm{best}}$ **então**
          $d_{\mathrm{best}} \leftarrow d_{T_i \times k}$
          $N_c \leftarrow k$
    $T^0_{i+1} \leftarrow N_c$
    $U_{N_c} \leftarrow 0$
  BuscaEmCoordenadas(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, m, c, r, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, θ, T^0, c, x^0)
  retorne T^0 e x^0

Depois de calculado o tour inicial, o processo iterativo inicia com as heurísticas de busca local. Os Algoritmos 5 e 6 descrevem a heurística 2-Opt adaptada ao CTSP. Ou seja, além de realizar uma busca local pelos vizinhos 2-Opt de um tour, para cada nova permutação obtida a partir de um movimento 2-Opt os pontos visitados são recalculados com o Algoritmo 3 para somente então

avaliar seu comprimento e verificar se houve uma melhora ou não em relação ao tour atual. Essa busca pelos vizinhos pode ser feita de acordo com duas estratégias distintas: *melhor vizinho* ou *primeiro vizinho* que melhora.

O Algoritmo 5, que utiliza a estratégia do *melhor vizinho*, recebe um tour inicial (T^0, x^0) construído com o Algoritmo 4, avalia o comprimento de todos os seus vizinhos 2-Opt com os pontos otimizados e o atualiza com o tour que apresentar o menor comprimento, obtendo assim o tour (T^1, x^1). Repetimos este processo com (T^1, x^1) gerando (T^2, x^2) e assim sucessivamente, de maneira iterativa, até obter um tour (T^ℓ, x^ℓ) para o qual não existem vizinhos 2-Opt, com a lógica acima, de comprimento menor que o de (T^ℓ, x^ℓ) ou até atingir um número máximo de iterações.

Algoritmo 5: Busca local que utiliza movimentos do tipo 2-opt e o método de busca em coordenadas para construir a vizinhança.

Entrada: $\alpha \in (0,1)$, $\beta > 0$, $0 < \sigma_1 \leq \sigma_2 < 1$, $\epsilon^{\mathrm{N}} > 0$, $\ell^{\mathrm{N}}_{\max} \geq 0$, m, $c = (c_1, \ldots, c_m)$, $r = (r_1, \ldots, r_m)$, $\epsilon^{\mathrm{BC}} > 0$, $\ell^{\mathrm{BC}}_{\max} \geq 0$, $\ell^{\mathrm{2opt}}_{\max} \geq 0$, $\theta = (\theta_1, \ldots, \theta_m)$, T^0, $x^0 = (x^0_1, \ldots, x^0_m)$.
Saída: T^*, $x^* = (x^*_1, \ldots, x^*_m)$.

Função 2optBest(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, $\ell^{\mathrm{2opt}}_{\max}$, m, c, r, T^0, x^0, θ, T^*, x^*)
  $\ell \leftarrow 0$
  faça
    $T^{\mathrm{best}} \leftarrow T^\ell$ e $x^{\mathrm{best}} \leftarrow x^\ell$
    para $i \leftarrow 1$ **até** $(m-2)$ **faça**
      para $j \leftarrow i+2$ **até** m **faça**
        $T^{\mathrm{2opt}}_k \leftarrow T^\ell_k$ para $k \in \{1, \ldots, i\} \cup \{j+1, \ldots, m\}$
        $T^{\mathrm{2opt}}_k \leftarrow T^\ell_{j-i+1-k}$ para $k \in \{i+1, \ldots, j\}$
        BuscaEmCoordenadas(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, m, c, r, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, θ, T^{2opt}, x^ℓ, x^{2opt})
        se $f(T^{\mathrm{2opt}}, x^{\mathrm{2opt}}) < f(T^{\mathrm{best}}, x^{\mathrm{best}})$ **então**
          $T^{\mathrm{best}} \leftarrow T^{\mathrm{2opt}}$ e $x^{\mathrm{best}} \leftarrow x^{\mathrm{2opt}}$
    $\delta \leftarrow f(T^\ell, x^\ell) - f(T^{\mathrm{best}}, x^{\mathrm{best}})$
    se $\delta > 0$ **então**
      $T^{\ell+1} := T^{\mathrm{best}}$ e $x^{\ell+1} := x^{\mathrm{best}}$
    $\ell \leftarrow \ell + 1$
  enquanto $\delta > 0$ e $\ell < \ell^{\mathrm{2opt}}_{\max}$
  $T^* := T^\ell$ e $x^* := x^\ell$
  retorne T^* e x^*

O Algoritmo 6, por sua vez, segue a lógica do algoritmo anterior, porém usa a estratégia do *primeiro vizinho* que melhora. Neste caso, o próximo iterando $(T^{\ell+1}, x^{\ell+1})$ recebe o primeiro vizinho 2-Opt com pontos otimizados que apresentar um comprimento menor que o tour atual.

Outros métodos heurísticos podem ser empregadas no lugar do 2-Opt para obter as permutações. É o caso da heurística de inserção abordada na Seção 2.1.2, que utiliza movimentos conhecidos por "realocação" para construir a vizinhança. O Algoritmo 7 utiliza a heurística de inserção com a

Algoritmo 6: Busca local que utiliza movimentos do tipo 2-opt e o método de busca em coordenadas para construir a vizinhança. Cada iterando é selecionado pela estratégia do primeiro vizinho que melhora.

Entrada: $\alpha \in (0,1)$, $\beta > 0$, $0 < \sigma_1 \leq \sigma_2 < 1$, $\epsilon^{\mathrm{N}} > 0$, $\ell^{\mathrm{N}}_{\max} \geq 0$, m, $c = (c_1, \ldots, c_m)$, $r = (r_1, \ldots, r_m)$, $\epsilon^{\mathrm{BC}} > 0$, $\ell^{\mathrm{BC}}_{\max} \geq 0$, $\ell^{\mathrm{2opt}}_{\max} \geq 0$, σ, $\theta = (\theta_1, \ldots, \theta_m)$, T^0, $x^0 = (x^0_1, \ldots, x^0_m)$.
Saída: T^*, $x^* = (x^*_1, \ldots, x^*_m)$.

Função 2optFirst(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, $\ell^{\mathrm{2opt}}_{\max}$, m, c, r, T^0, x^0, θ, T^*, x^*)
    $\ell \leftarrow 0$
    faça
        $\sigma \leftarrow 1$
        $i \leftarrow 1$
        enquanto $i \leq m-2$ e $\sigma \geq 1$ **faça**
            $j \leftarrow i+2$
            enquanto $j \leq m$ e $\sigma \geq 1$ **faça**
                $T^{\mathrm{2opt}}_k \leftarrow T^{\ell}_k$ para $k \in \{1, \ldots, i\} \cup \{j+1, \ldots, m\}$
                $T^{\mathrm{2opt}}_k \leftarrow T^{\ell}_{j-i+1-k}$ para $k \in \{i+1, \ldots, j\}$
                BuscaEmCoordenadas(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, m, c, r, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, θ, T^{2opt}, x^{ℓ}, x^{2opt})
                se $f(T^{\mathrm{2opt}}, x^{\mathrm{2opt}}) < f(T^{\ell}, x^{\ell})$ **então**
                    $\sigma \leftarrow 0$
                $j \leftarrow j+1$
            $i \leftarrow i+1$
        $\delta \leftarrow f(T^{\ell}, x^{\ell}) - f(T^{\mathrm{2opt}}, x^{\mathrm{2opt}})$
        se $\delta > 0$ **então**
            $T^{\ell+1} := T^{\mathrm{2opt}}$ e $x^{\ell+1} := x^{\mathrm{2opt}}$
        $\ell \leftarrow \ell+1$
    enquanto $\delta > 0$ e $\ell < \ell^{\mathrm{2opt}}_{\max}$
    $T^* := T^{\ell}$ e $x^* := x^{\ell}$
    retorne T^* e x^*

estratégia do melhor vizinho e, assim como nos métodos anteriores, o Algoritmo 3 de busca em coordenadas para construir a vizinhança de cada tour. Relembrando, na estratégia do melhor vizinho, todos os vizinhos são avaliados e o melhor vizinho, isto é, o tour com menor custo da função objetivo em relação aos demais, é escolhido como novo iterando.

Por outro lado, o Algoritmo 8 considera o primeiro vizinho obtido pela heurística de inserção e busca em coordenadas que melhora o tour atual e o define como novo iterando.

O capítulo seguinte é dedicado aos experimentos numéricos realizados com os algoritmos abordados nesta seção para o problema do caixeiro viajante contínuo (3.1).

Algoritmo 7: Busca local que utiliza movimentos de realocação, a estratégia do melhor movimento e o método de busca em coordenadas para construir a vizinhança.

Entrada: $\alpha \in (0,1)$, $\beta > 0$, $0 < \sigma_1 \leq \sigma_2 < 1$, $\epsilon^{\mathrm{N}} > 0$, $\ell^{\mathrm{N}}_{\max} \geq 0$, m, $c = (c_1, \ldots, c_m)$, $r = (r_1, \ldots, r_m)$, $\epsilon^{\mathrm{BC}} > 0$, $\ell^{\mathrm{BC}}_{\max} \geq 0$, $\ell_{\max} \geq 0$, $\theta = (\theta_1, \ldots, \theta_m)$, T^0, $x^0 = (x^0_1, \ldots, x^0_m)$.

Saída: T^*, $x^* = (x^*_1, \ldots, x^*_m)$.

Função InsercaoBest(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, $\ell_{\max}$, m, c, r, T^0, x^0, θ, T^*, x^*)
- $\ell \leftarrow 0$
- **faça**
  - $T^{\mathrm{best}} \leftarrow T^\ell$ e $x^{\mathrm{best}} \leftarrow x^\ell$
  - **para** $i \leftarrow 1$ **até** m **faça**
    - **para** $j \leftarrow 1$ **até** m **faça**
      - **se** $j \neq i$ **então**
        - **se** $j < i$ **então**
          - $T^{\mathrm{new}}_k \leftarrow T^\ell_k$ para $k \in \{1, \ldots, j-1\} \cup \{i+1, \ldots, m\}$
          - $T^{\mathrm{new}}_j \leftarrow T^\ell_i$
          - $T^{\mathrm{new}}_k \leftarrow T^\ell_{k-1}$ para $k \in \{j+1, \ldots, i\}$
        - **senão**
          - $T^{\mathrm{new}}_k \leftarrow T^\ell_k$ para $k \in \{1, \ldots, i-1\} \cup \{j+1, \ldots, m\}$
          - $T^{\mathrm{new}}_j \leftarrow T^\ell_i$
          - $T^{\mathrm{new}}_k \leftarrow T^\ell_{k+1}$ para $k \in \{i, \ldots, j-1\}$
        - BuscaEmCoordenadas(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, m, c, r, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, θ, T^{new}, x^ℓ, x^{new})
        - **se** $f(T^{\mathrm{new}}, x^{\mathrm{new}}) < f(T^{\mathrm{best}}, x^{\mathrm{best}})$ **então**
          - $T^{\mathrm{best}} \leftarrow T^{\mathrm{new}}$ e $x^{\mathrm{best}} \leftarrow x^{\mathrm{new}}$
    - $\delta \leftarrow f(T^\ell, x^\ell) - f(T^{\mathrm{best}}, x^{\mathrm{best}})$
    - **se** $\delta > 0$ **então**
      - $T^{\ell+1} := T^{\mathrm{best}}$ e $x^{\ell+1} := x^{\mathrm{best}}$
    - $\ell \leftarrow \ell + 1$
  - **enquanto** $\delta > 0$ e $\ell < \ell_{\max}$
  - $T^* := T^\ell$ e $x^* := x^\ell$
  - **retorne** T^* e x^*

Algoritmo 8: Busca local que utiliza movimentos de realocação, a estratégia do primeiro melhor movimento e o método de busca em coordenadas para construir a vizinhança.

Entrada: $\alpha \in (0,1)$, $\beta > 0$, $0 < \sigma_1 \leq \sigma_2 < 1$, $\epsilon^{\mathrm{N}} > 0$, $\ell^{\mathrm{N}}_{\max} \geq 0$, m, $c = (c_1, \ldots, c_m)$, $r = (r_1, \ldots, r_m)$, $\epsilon^{\mathrm{BC}} > 0$, $\ell^{\mathrm{BC}}_{\max} \geq 0$, $\ell_{\max} \geq 0$, σ, $\theta = (\theta_1, \ldots, \theta_m)$, T^0, $x^0 = (x^0_1, \ldots, x^0_m)$.
Saída: T^*, $x^* = (x^*_1, \ldots, x^*_m)$.

Função InsercaoFirst(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, $\ell_{\max}$, m, c, r, T^0, x^0, θ, T^*, x^*)
    $\ell \leftarrow 0$
    faça
        $\sigma \leftarrow 1$
        $i \leftarrow 1$
        enquanto $i \leq m$ *e* $\sigma \geq 1$ **faça**
            $j \leftarrow 1$
            enquanto $j \leq m$ *e* $\sigma \geq 1$ **faça**
                se $j \neq i$ **então**
                    se $j < i$ **então**
                        $T^{\mathrm{new}}_k \leftarrow T^{\ell}_k$ para $k \in \{1, \ldots, j-1\} \cup \{i+1, \ldots, m\}$
                        $T^{\mathrm{new}}_j \leftarrow T^{\ell}_i$
                        $T^{\mathrm{new}}_k \leftarrow T^{\ell}_{k-1}$ para $k \in \{j+1, \ldots, i\}$
                    senão
                        $T^{\mathrm{new}}_k \leftarrow T^{\ell}_k$ para $k \in \{1, \ldots, i-1\} \cup \{j+1, \ldots, m\}$
                        $T^{\mathrm{new}}_k \leftarrow T^{\ell}_{k+1}$ para $k \in \{i, \ldots, j-1\}$
                        $T^{\mathrm{new}}_j \leftarrow T^{\ell}_i$
                    BuscaEmCoordenadas(α, β, σ_1, σ_2, ϵ^{N}, $\ell^{\mathrm{N}}_{\max}$, m, c, r, ϵ^{BC}, $\ell^{\mathrm{BC}}_{\max}$, θ, T^{new}, x^{ℓ}, x^{new})
                    se $f(T^{\mathrm{new}}, x^{\mathrm{new}}) < f(T^{\ell}, x^{\ell})$ **então**
                        $\sigma \leftarrow 0$
                    $j \leftarrow j + 1$
                $i \leftarrow i + 1$
            $\delta \leftarrow f(T^{\ell}, x^{\ell}) - f(T^{\mathrm{new}}, x^{\mathrm{new}})$
            se $\delta > 0$ **então**
                $T^{\ell+1} := T^{\mathrm{new}}$ e $x^{\ell+1} := x^{\mathrm{new}}$
            $\ell \leftarrow \ell + 1$
    enquanto $\delta > 0$ e $\ell < \ell_{\max}$
    $T^* := T^{\ell}$ e $x^* := x^{\ell}$
    retorne T^* e x^*

Capítulo 4

Experimentos Numéricos

Neste capítulo analisamos o desempenho dos algoritmos descritos no capítulo anterior para o problema do caixeiro viajante contínuo (3.1). Iniciamos o capítulo com experimentos que tem o intuito de demonstrar a atuação da heurística do vizinho mais próximo (NN) na construção de uma permutação inicial e do método de busca em bloco de coordenadas, dado pelo Algoritmo 3, na otimização dos pontos visitados. Por conseguinte, os algoritmos são avaliados em um conjunto de instâncias geradas de maneira aleatória para diferentes quantidades de bolas. No terceiro grupo de experimentos, os algoritmos são testados em algumas instâncias da biblioteca TSPLIB [28] para o TSP Euclidiano no $\mathbb{R}^2$. Neste caso, as instâncias são adaptadas para se encaixar na formulação dada pelo problema (3.1). Para a implementação dos algoritmos utilizamos a linguagem de programação `Fortran 90` e como compilador `GFortran`. Os experimentos foram realizados em um computador com processador Intel Core i7-8700 CPU @ 3.20GHz e 16GB de RAM. As figuras foram construídas com o `MetaPost` e para os gráficos utilizamos a linguagem de programação `Python 3` com a biblioteca `Matplotlib`.

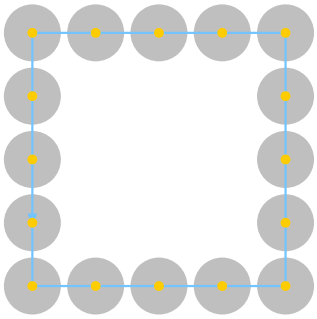
4.1 Experimentos iniciais

Nos experimentos que seguem, cada instância I fornecida ao programa é formada por um conjunto de m pontos $c^i \in \mathbb{R}^2$ que correspondem aos centros das bolas $\mathcal{B}_i$. A determinação dos raios r^i ocorre de acordo com o seguinte processo: primeiro, avaliamos a distância Euclidiana entre o centro c^i e todos os demais pontos $\{c^1, \ldots, c^{i-1}, c^{i+1}, \ldots, c^m\}$. Em seguida, selecionamos a menor delas, digamos d_i. Com isso, definimos o raio r^i associado ao centro c^i através da fórmula

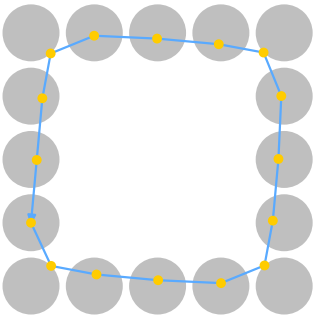
$$r^i := \alpha \left(\frac{d_i}{2} \right), \tag{4.1}$$

com $\alpha \in (0{,}0, 0{,}9)$. Desse modo, as bolas obtidas terão intersecção vazia e os raios poderão ser de medidas variadas. Nos experimentos a seguir utilizamos $\alpha = 0{,}9$.

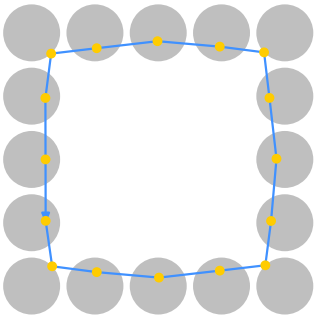
Para o primeiro conjunto de experimentos, os centros das bolas de cada instância foram escolhidos com o objetivo de possibilitar uma visualização mais detalhada da atuação dos algoritmos do capítulo anterior, sobretudo dos Algoritmos 3 e 4. A heurística NN, como visto no Capítulo 2, constrói um tour começando de um ponto qualquer e adicionando ao tour o ponto mais próximo do atual que ainda não foi visitado. Para o problema do caixeiro viajante contínuo (3.1), esta heurística é empregada no Algoritmo 4 para construir a ordenação do tour inicial utilizando os centros das bolas e a distância entre eles como referência. A Figura 4.1 apresenta o processo de construção do tour inicial pelo Algoritmo 4 para uma instância com 16 bolas de raios uniformes. Nas figuras a seguir a notação "u.c." denota "unidades de comprimento".



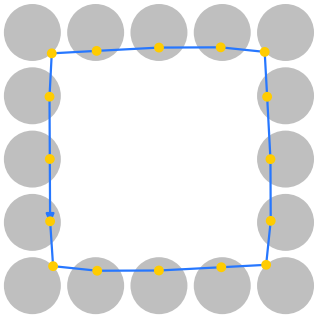
(a) Ciclo: 0. Comprimento: 16,0 u.c.



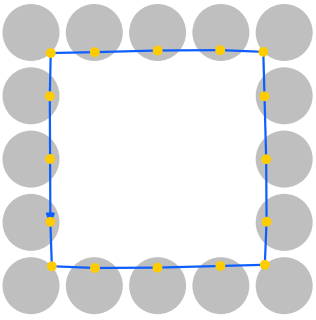
(b) Ciclo: 1. Comprimento: 13,762 u.c.



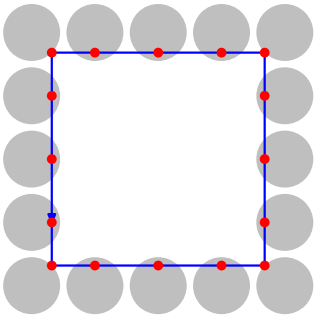
(c) Ciclo: 2. Comprimento: 13,531 u.c.



(d) Ciclo: 3. Comprimento: 13,476 u.c.



(e) Ciclo: 4. Comprimento: 13,459 u.c.



(f) Ciclo: 14. Tour ótimo: 13,454 u.c.

Figura 4.1: Evolução dos tours a cada ciclo do método BDC (3).

A Figura 4.1a é obtida aplicando apenas a heurística NN, que define a ordem T^0 com a qual as bolas devem ser percorridas no tour inicial. Em seguida, o Algoritmo 3 de busca em bloco de coordenadas (BCD) entra em ação, recalculando todos os pontos do tour para diminuir seu comprimento. Neste algoritmo, cada ponto é recalculado seguindo a ordem dada por T^0 e dizemos

que um ciclo é completado somente após todos os pontos serem atualizados. Como estabelecido no Algoritmo 3, o algoritmo para quando o módulo da diferença entre os custos do tour do ciclo atual e do anterior for menor que um certo ϵ^{BC} ou quando atingir um número máximo de ciclos $\ell^{\mathrm{BC}}_{\max}$. Para os experimentos, definimos $\epsilon^{\mathrm{bc}} := 10^{-8}$ e $\ell^{\mathrm{BC}}_{\max} := 200$. No exemplo da Figura 4.1, o critério de parada foi satisfeito no ciclo 14, completando a obtenção do tour inicial (T^0, x^0) pelo Algoritmo 4. Neste exemplo em específico, o Algoritmo 4 já foi suficiente para obter uma solução, de modo que o tour (T^0, x^0) dado na Figura 4.1f corresponde à solução ótima do problema, pois qualquer reordenação na ordem de visitação das bolas resultaria em tours de custo maior.

Na Figura 4.2 temos outro exemplo que demonstra a atuação do método BCD dado pelo Algoritmo 3. Neste caso, os pontos em amarelo calculados a cada ciclo foram condensados em uma mesma figura para podermos visualizar suas trajetórias convergindo até os pontos em vermelho, que correspondem aos pontos do tour final obtido com 25 ciclos. As linhas em azul denotam a rota obtida em cada ciclo do Algoritmo 3 ao se percorrer os pontos em amarelo com a ordem dada pela heurística construtiva NN. A tonalidade dessas rotas vai se tornando mais escura conforme se aproxima da rota azul escura do ciclo 25.

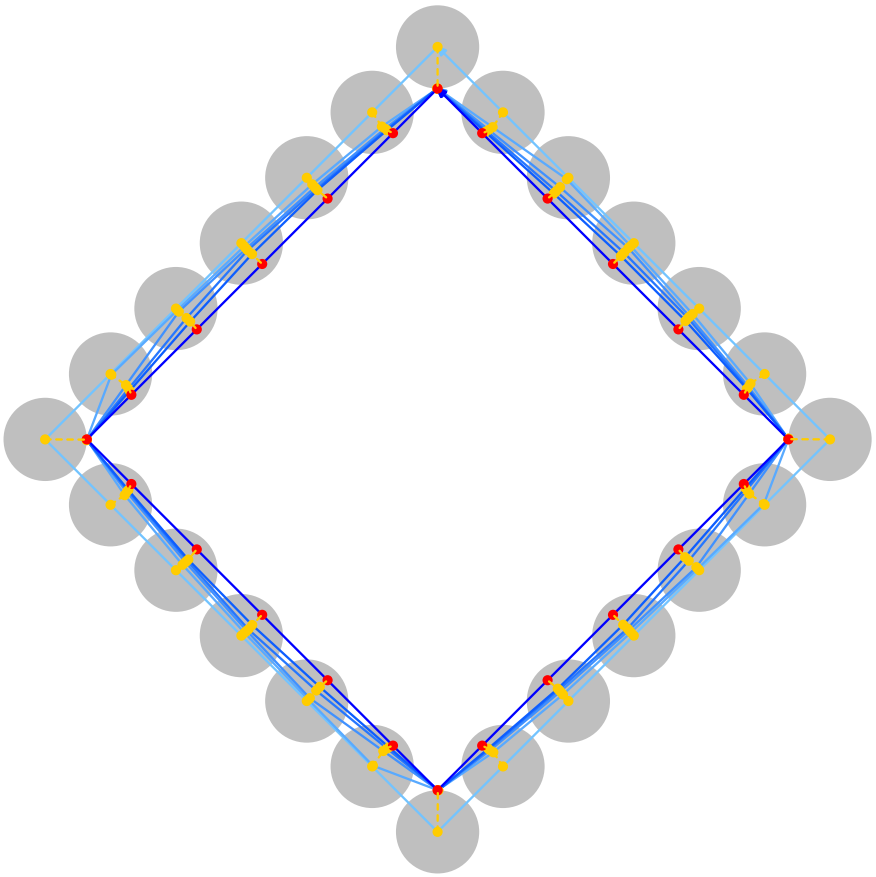
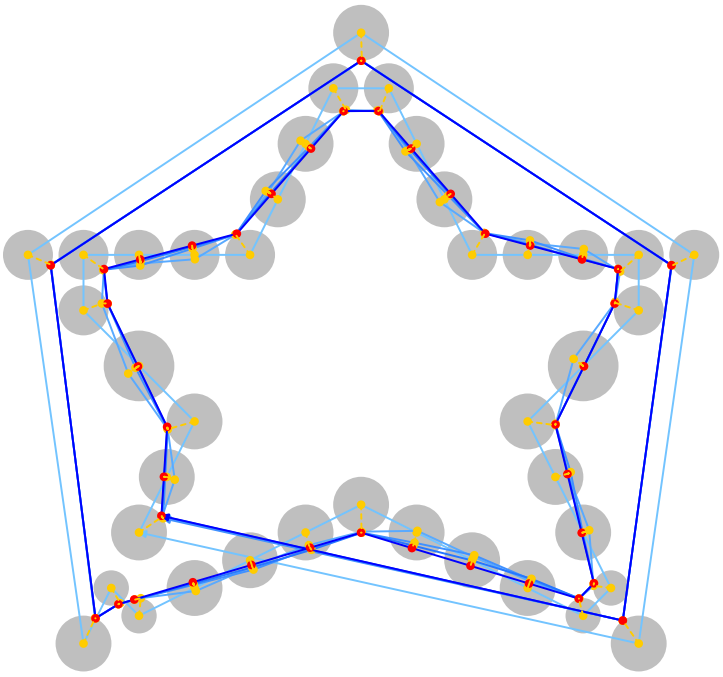


Figura 4.2: Evolução dos tours a cada ciclo do Algoritmo 3 para uma instância de 24 bolas com raios uniformes.



Ciclos	Comprimento do tour
0	76,2156
1	66,9183
2	66,5048
3	66,4456
4	66,4360
5	66,4341
⋮	⋮
15	66,4334

Figura 4.3: Construção do tour inicial pelo Algoritmo 4 para a instância com 40 bolas.

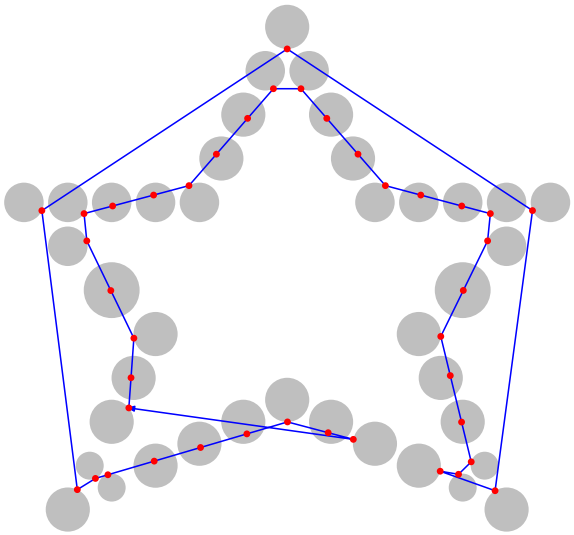
O comprimento total do tour da Figura 4.2 antes de atualizar os pontos pelo método BCD era de 33,94 u.c. Após 25 ciclos do método, esse custo caiu para 30,34 u.c. Ademais, na atualização dos pontos a cada ciclo, cerca de 20 deles são calculados pela estratégia da projeção dada pelo Algoritmo 1, enquanto que os demais são obtidos pelo método de Newton (2).

Para concluir esse primeiro conjunto de experimentos, consideramos uma instância com 40 bolas cujos centros foram escolhidos de modo que a disposição das bolas formasse a figura de uma estrela. O aprimoramento dos tours no processo de construção do tour inicial (T^0, x^0) é apresentado na Figura 4.3. Neste exemplo, o critério de parada do método BCD é atingido no ciclo 15, com a rota do tour (T^0, x^0) desenhada na cor azul escura com os pontos em vermelho. A figura e a tabela também apresentam os tours, e seus respectivos comprimentos, para 7 ciclos completos. Vale ressaltar que na construção da Figura 4.3 consideramos a permutação dada pela heurística NN e os pontos gerados pelo método BCD a partir dos centros da bolas. Uma abordagem alternativa seria escolher pontos aleatórios dentro de cada bola e então gerar o tour inicial a partir deles com o Algoritmo 4.

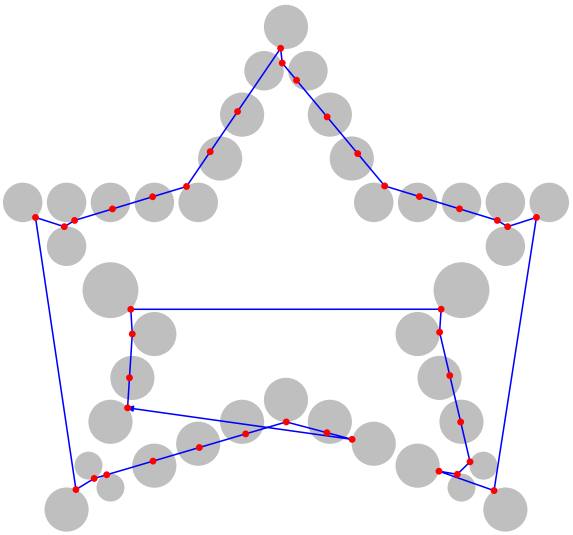
Observe que, apesar do método levar 15 ciclos, o decréscimo mais significativo no custo do tour ocorre no primeiro ciclo, com o custo variando de 76,2156 u.c. para 66,9183 u.c. Ademais, a Figura 4.3 também evidencia uma característica da heurística do vizinho mais próximo que dificulta a obtenção de soluções ótimas por ela. Como esta heurística atua percorrendo os pontos de acordo com a sua proximidade, é comum que ao final do processo sobrem pontos tais que sua inclusão no tour resulte num grande acréscimo para o custo total. É o que ocorre neste exemplo, em que

as bolas das pontas da estrela não são incorporadas “no meio” da construção do tour e assim, a inclusão delas ao final do processo resulta na perda da qualidade do tour inicial.

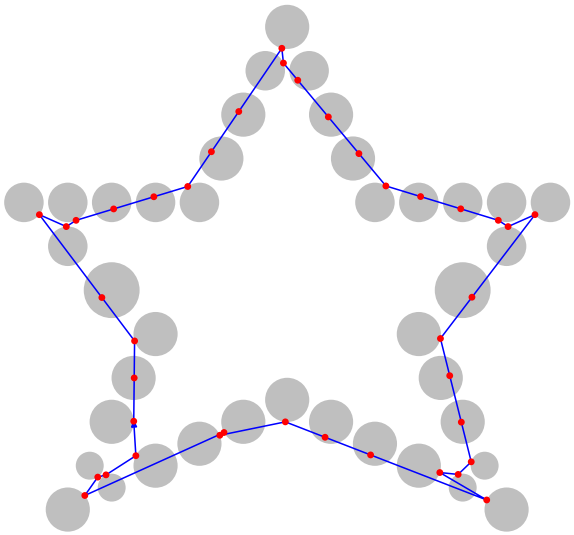
Por fim, apresentamos a solução obtida pelo método 2optBest para a instância de 40 bolas. Na Figura 4.4 é possível observar alguns tours gerados ao longo do processo iterativo do Algoritmo 5. As Figuras 4.4a a 4.4c representam os tours correspondentes aos iterandos 1, 5 e 10, respectivamente, enquanto que a Figura 4.4d correspondem ao tour do último iterando, obtido na iteração 20. O comprimento de cada tour é exibido abaixo das respectivas figuras.



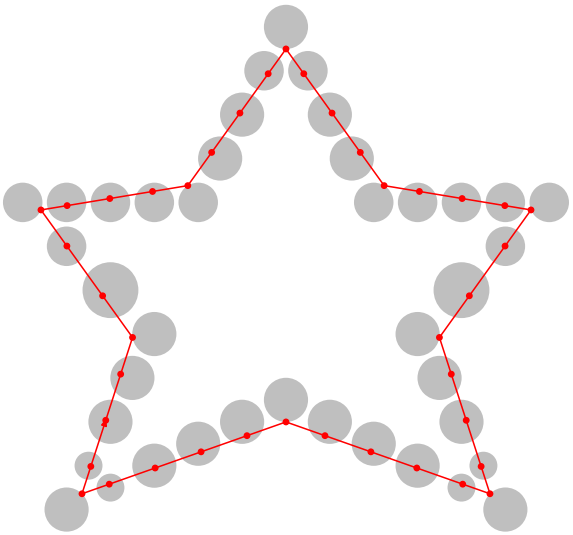
(a) 62,27 u.c. Iteração: 1.



(b) 54,45 u.c. Iteração: 5.



(c) 41,30 u.c. Iteração: 10.



(d) 38,95 u.c. Iteração: 20.

Figura 4.4: Amostra de tours obtidos pelo método 2optBest para a instância de 40 bolas.

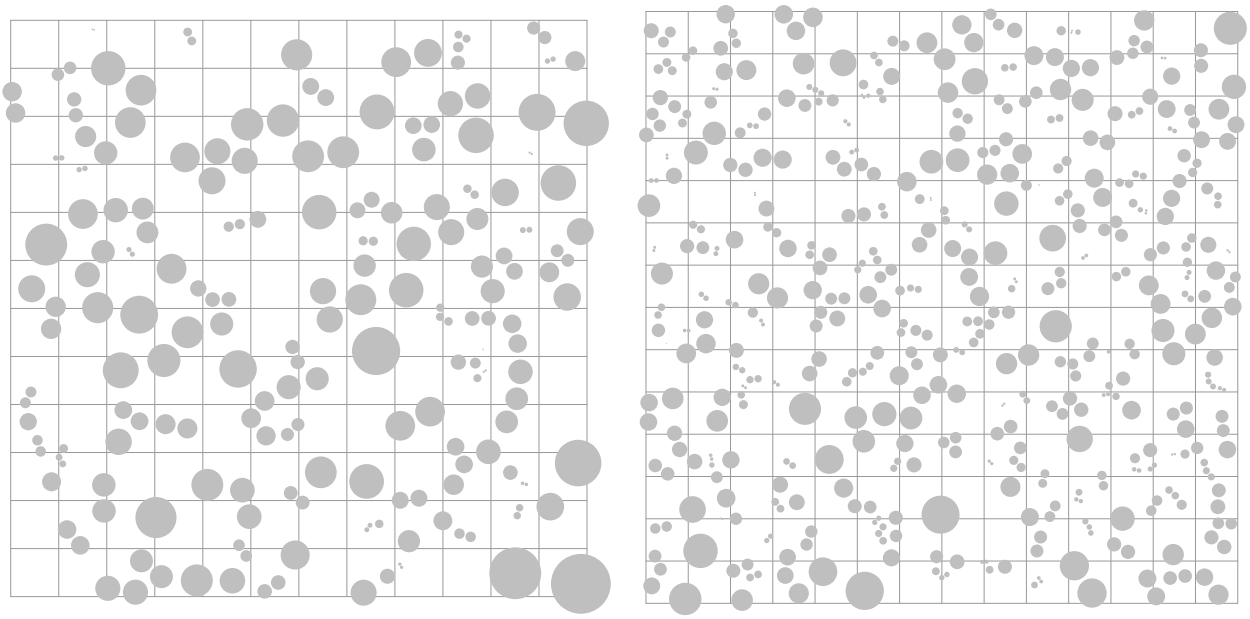
O Algoritmo 5 combina a heurística de melhoria 2-Opt e o método BCD para encontrar tours de menor custo a partir do tour atual através de movimentos do tipo 2-Opt e otimização dos pontos

visitados. Uma característica marcante desta heurística é que ela tende a desfazer as arestas que se "cruzam", reconectando os pontos de seus extremos de modo a evitá-las. No exemplo da estrela, este método consegue reordenar a visitação das bolas até a iteração 20. A partir do tour obtido nesta iteração não é mais possível obter tours melhores com as estratégias empregadas pelo Algoritmo 5.

4.2 Experimentos com instâncias aleatórias

Para o conjunto de experimentos desta seção, 10 instâncias aleatórias foram geradas para cada número de bolas $m \in \{50, 100, 200, 300, 400, 500\}$. Neste contexto, cada instância I é formada por um conjunto de m pares da forma (c^i, r^i) tais que $c^i \in \mathbb{R}^2$ e $r^i \in \mathbb{R}$ correspondem ao centro e raio da bola $\mathcal{B}_i$, respectivamente. As coordenadas dos pontos c^i são geradas de forma aleatória em um quadrado de lado 10 u.c. para $m = 50$ e 100, lado 12 u.c. para $m = 200$ e 300 e lado 14 u.c. para $m = 400$ e 500. Os raios r^i, por sua vez, são gerados pelo mesmo processo descrito no início da Seção 4.1, o qual garante que as bolas sejam disjuntas.

A Figura 4.5 a seguir apresenta como exemplo duas instâncias com diferentes números de bolas geradas de acordo com a estratégia descrita acima.



(a) Instância com 200 bolas.

(b) Instância com 500 bolas.

Figura 4.5: Exemplos de instâncias para o CTSP.

Iniciamos avaliando a implementação dos quatro algoritmos do capítulo anterior em um conjunto de 10 instâncias aleatórias com 100 bolas cada. A performance dos algoritmos é resumida na Tabela 4.1. Esta tabela é dividida em quatro porções, cada uma detalhando a implementação de um dos quatro métodos, 2optBest (5), 2optFirst (6), InsercaoBest (7) e InsercaoFirst (8), para as

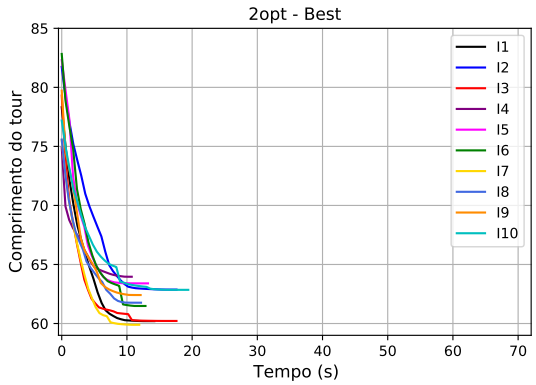
10 instâncias. Para cada instância e método a tabela contém, nesta ordem, o número de iterações que o algoritmo levou até satisfazer o critério de parada, o comprimento do tour final, o número total de vezes que o método BCD foi chamado e o total de ciclos realizados por ele, assim como uma média de chamadas e ciclos do método BCD por iteração do algoritmo. A tabela também apresenta uma média de quantos pontos, a cada ciclo do BCDM, foram calculados utilizando o método de Newton e quantos foram obtidos com a estratégia da projeção proposta no Algoritmo 1. A última coluna exibe o tempo, em segundos, de execução do algoritmo para cada instância.

Instância		#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
				#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
2optBest	I1	23	60,2042315360	111.573	745.154	4.851,0	32.398,0	6,7	71,78	28,22	14,21
	I2	29	62,8704034539	140.679	943.523	4.851,0	32.535,3	6,7	69,81	30,19	17,60
	I3	30	60,2177062020	145.530	919.551	4.851,0	30.651,7	6,3	74,65	25,35	17,60
	I4	18	63,9611905511	87.318	552.535	4.851,0	30.696,4	6,3	71,30	28,70	10,74
	I5	22	63,3996293053	106.722	686.148	4.851,0	31.188,5	6,4	73,04	26,96	13,23
	I6	21	61,4855291810	101.871	665.328	4.851,0	31.682,3	6,5	72,04	27,96	12,85
	I7	20	59,8920966305	97.020	613.434	4.851,0	30.671,7	6,3	74,78	25,22	11,89
	I8	20	61,7622437286	97.020	646.046	4.851,0	32.302,3	6,7	66,15	33,85	12,16
	I9	19	62,4119445480	92.169	634.969	4.851,0	33.419,4	6,9	69,81	30,19	12,10
	I10	32	62,8519958103	155.232	1.025.800	4.851,0	32.056,2	6,6	70,55	29,45	19,41
Média		23,4	61,9056970947	113.513,4	743.248,8	4.851,0	31.760,2	6,5	71,39	28,61	14,18
2optFirst	I1	86	59,2537681881	134.247	887.457	1.561,0	10.319,3	6,6	68,97	31,03	16,71
	I2	66	64,2694670071	78.522	516.636	1.189,7	7.827,8	6,6	67,32	32,68	9,80
	I3	87	64,2640012093	104.150	717.627	1.197,1	8.248,6	6,9	74,05	25,95	13,63
	I4	115	61,2672304577	140.383	891.568	1.220,7	7.752,8	6,4	71,08	28,92	16,81
	I5	96	65,7551477403	225.516	1.431.076	2.349,1	14.907,0	6,3	70,80	29,20	26,63
	I6	97	62,2965137809	158.237	1.043.334	1.631,3	10.756,0	6,6	72,15	27,85	19,86
	I7	83	61,8153757611	90.737	565.136	1.093,2	6.808,9	6,2	70,73	29,27	10,85
	I8	68	61,6956322584	72.518	455.297	1.066,4	6.695,5	6,3	66,86	33,14	8,76
	I9	64	65,7874371987	116.434	841.886	1.819,3	13.154,5	7,2	71,06	28,94	15,79
	I10	134	65,7065246469	142.650	1.012.137	1.064,6	7.553,3	7,1	68,36	31,64	18,94
Média		89,6	63,2111098248	126.339,4	836.215,4	1.419,3	9.402,4	6,6	70,14	29,86	15,78
InsercaoBest	I1	37	71,1662738477	366.300	2.472.501	9.900,0	66.824,4	6,7	70,31	29,69	44,99
	I2	42	66,2064457580	415.800	2.968.203	9.900,0	70.671,5	7,1	67,44	32,56	52,42
	I3	48	68,5834080002	475.200	3.300.646	9.900,0	68.763,5	6,9	68,84	31,16	58,79
	I4	30	69,2318146780	297.000	1.963.616	9.900,0	65.453,9	6,6	68,32	31,68	35,61
	I5	29	68,9386239661	287.100	1.912.330	9.900,0	65.942,4	6,7	71,83	28,17	35,17
	I6	40	70,2823793770	396.000	2.621.160	9.900,0	65.529,0	6,6	72,26	27,74	47,81
	I7	29	66,2907660198	287.100	1.871.200	9.900,0	64.524,1	6,5	72,36	27,64	34,10
	I8	38	63,1585409012	376.200	2.549.391	9.900,0	67.089,2	6,8	67,93	32,07	45,63
	I9	42	71,1590802459	415.800	2.860.809	9.900,0	68.114,5	6,9	71,40	28,60	51,87
	I10	34	64,5909863283	336.600	2.304.266	9.900,0	67.772,5	6,8	69,77	30,23	42,05
Média		36,9	67,9608319122	365.310,0	2.482.412,2	9.900,0	67.068,5	6,8	70,04	29,96	44,84
InsercaoFirst	I1	72	70,5897392582	123.583	910.015	1.716,4	12.639,1	7,4	68,09	31,91	17,12
	I2	103	67,2948366466	383.216	2.751.884	3.720,5	26.717,3	7,2	64,22	35,78	47,75
	I3	152	68,0670781551	241.455	1.611.897	1.588,5	10.604,6	6,7	65,82	34,18	29,02
	I4	56	69,2387310636	176.929	1.257.568	3.159,4	22.456,6	7,1	69,17	30,83	23,06
	I5	132	72,3164843685	208.727	1.381.384	1.581,3	10.465,0	6,6	72,06	27,94	25,49
	I6	93	70,9170627910	270.010	1.905.164	2.903,3	20.485,6	7,1	67,80	32,20	34,38
	I7	82	66,1548207026	240.757	1.734.301	2.936,1	21.150,0	7,2	70,43	29,57	31,17
	I8	119	61,5502871482	189.393	1.196.243	1.591,5	10.052,5	6,3	70,87	29,13	22,34
	I9	99	69,3535431876	228.298	1.637.978	2.306,0	16.545,2	7,2	66,90	33,10	29,62
	I10	127	63,5653800995	183.869	1.234.533	1.447,8	9.720,7	6,7	71,12	28,88	23,19
Média		103,5	67,9047963421	224.623,7	1.562.096,7	2.295,1	16.083,7	6,9	68,65	31,35	28,32

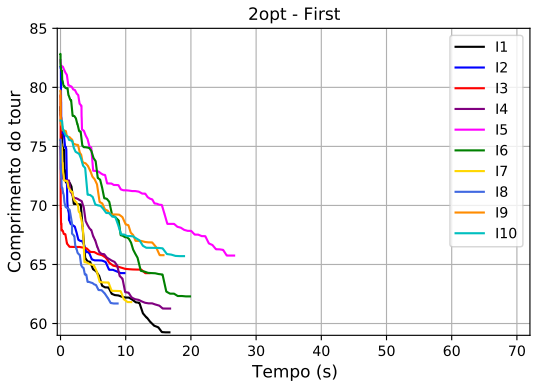
Tabela 4.1: Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 100$ bolas.

A Tabela 4.1 também apresenta a média aritmética entre os valores de cada coluna para cada algoritmo. Uma comparação inicial entre os algoritmos pode ser feita analisando o comprimento do tour final construído para cada instância e o número de iterações necessárias para obtê-lo. Para tanto, a Figura 4.6 apresenta, para cada algoritmo, o gráfico do tempo (em segundos) pelo

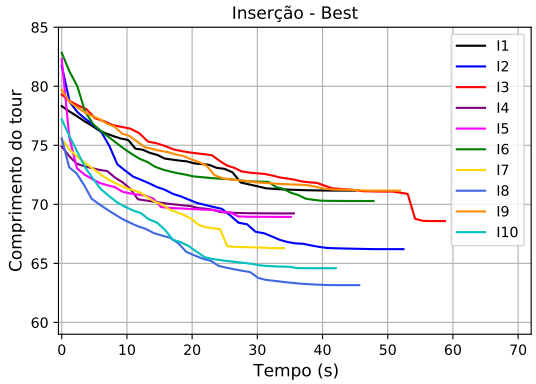
comprimento do tour para cada uma das 10 instâncias.



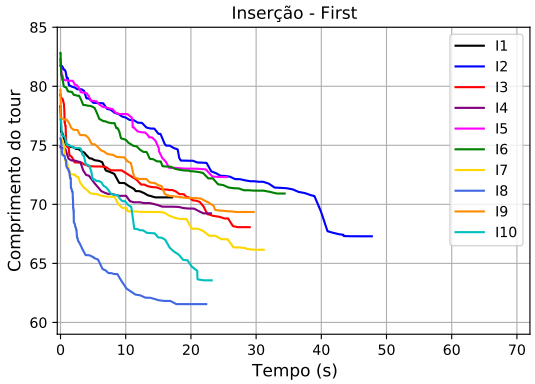
(a) 2optBest para as 10 instâncias.



(b) 2optFirst para as 10 instâncias.



(c) InsercaoBest para as 10 instâncias.



(d) InsercaoFirst para as 10 instâncias.

Figura 4.6: Gráficos do comprimento do tour em função do tempo, em segundos, obtidos pelos métodos quando aplicados às instâncias com 100 bolas.

Comparando o gráfico da Figura 4.6a com os demais, temos que o Algoritmo 5 do método 2optBest consegue encontrar tours melhores e em tempo menor que os outros métodos, sobretudo quando comparado aos Algoritmos 7 e 8 de inserção. De modo geral, para as instâncias com 100 bolas, os tours encontrados pelos algoritmos de inserção não são atrativos, apresentando na maioria dos casos um custo maior que o tour encontrado pelos algoritmos 2opt e mais tempo para serem computados, com o algoritmo de InsercaoBest levando em média três vezes mais tempo que o algoritmo 2optBest na construção de uma solução. Um dos motivos pelo qual os métodos de inserção exigem maior tempo de execução está no fato de realizar mais chamadas do método BCD a cada iteração, cerca de duas vezes mais. Isso se deve ao fato de que a cada iteração do método, o número de movimentos de realocação que o método de inserção pode fazer é aproximadamente o dobro da quantidade de movimentos do tipo 2-Opt que os métodos 2opt realizam. Logo, quando a estratégia do melhor vizinho é empregada, o algoritmo InsercaoBest tem, a cada iteração, o dobro de vizinhos para avaliar em relação ao algoritmo 2optBest.

Outra informação interessante que pode ser extraída da Tabela 4.1 diz respeito a quantidade de ciclos que o método BCD realiza quando é chamado. De acordo com a tabela, independente do algoritmo empregado e da instância, o método de busca em bloco de coordenadas do Algoritmo 3 executa em média de 6 a 7 ciclos por chamada. Além disso, a maioria dos pontos, em geral mais da metade, é calculada pelo método de Newton a cada ciclo.

As figuras a seguir apresentam com maiores detalhes a atuação dos métodos para a instância I2. O Figura 4.7 mostra as curvas obtidas por cada método quando comparamos seu tempo de execução e o comprimento do tour. Assim como observado para as demais instâncias, o tour de menor comprimento, 62,8704 u.c., é obtido pelo algoritmo 2optBest. Em contrapartida, o menor tempo, 9,8s, é alcançado pelo 2optFirst.

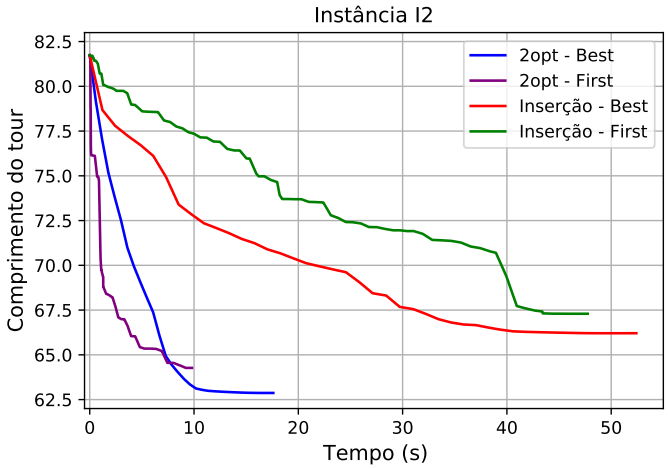
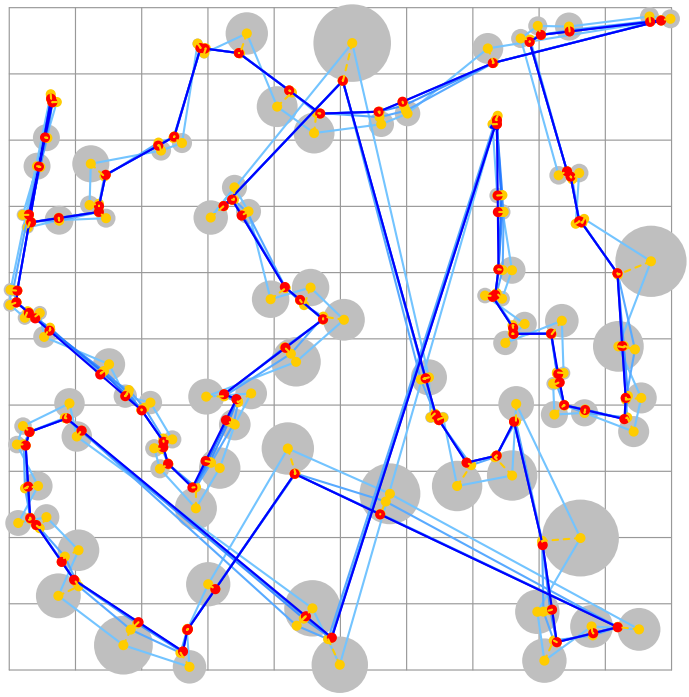


Figura 4.7: Comprimento do tour em função do tempo de cada método para a instância I2 de 100 bolas.

Para todos os quatro métodos de melhoria, o tour inicial para a instância I2 é construído com o Algoritmo 4. Ele pode ser visualizado na Figura 4.8. Assim como nos exemplos da Seção 4.1, os pontos iniciais estão em amarelo e mudam para vermelho no ciclo 17. A evolução destes pontos é demarcada com uma linha tracejada também na cor amarela. As linhas contínuas em azul claro denotam a rota de cada ciclo do método BCD, e vão se tornando mais escuras com o decorrer dos ciclos até atingir a tonalidade azul escura no ciclo 17. As mudanças mais significativas ocorrem nos ciclos iniciais, principalmente no ciclo 1, com um decréscimo de 16,96 u.c. no comprimento do tour. Nos demais ciclos, ocorrem apenas pequenos ajustes na localização dos pontos, com o comprimento do tour variando de 81,7380 u.c. ao final do ciclo 5 para 81,7372 u.c. ao final do ciclo 17.

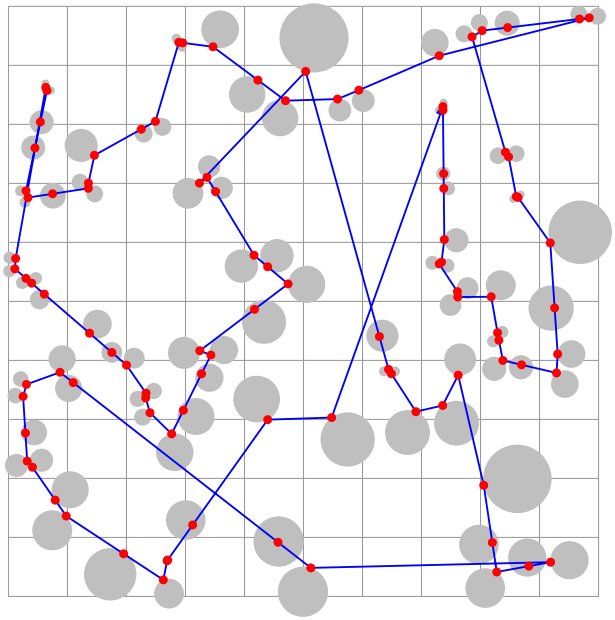
Agora, cada um dos algoritmos de melhoria, 2optBest e First e InsercaoBest e First, podem ser aplicados para melhorar o tour inicial representado na Figura 4.8. Algumas iterações de cada um destes métodos são apresentadas nas figuras a seguir.



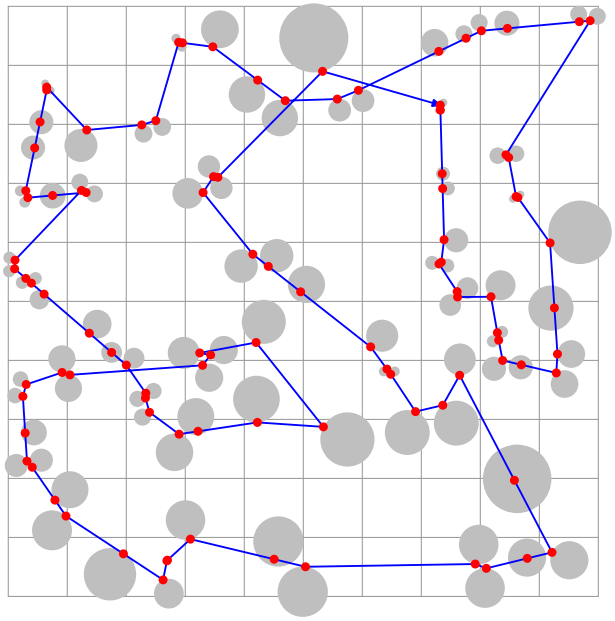
Ciclos	Comprimento do tour
0	99,2087
1	82,2424
2	81,7832
3	81,7451
4	81,7394
5	81,7380
⋮	⋮
17	81,7372

Figura 4.8: Construção do tour inicial para a instância I2 de 100 bolas.

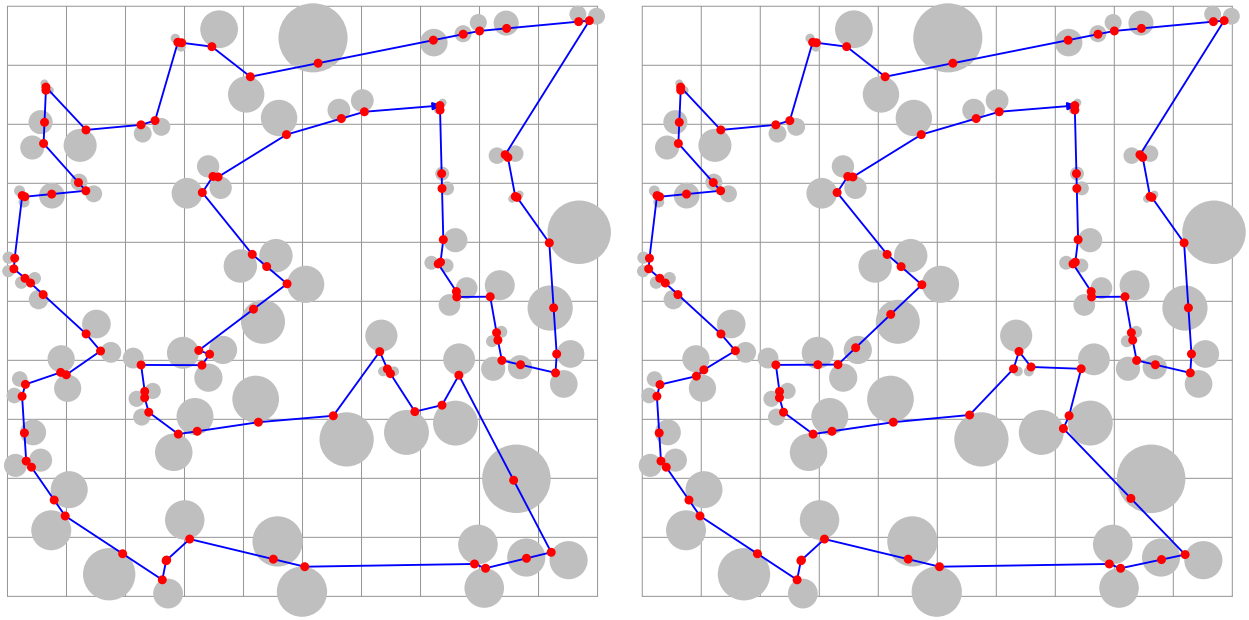
A Algoritmo 5 apresenta, respectivamente, as iterações 1, 10, 15 e 20 do Algoritmo 5 do método 2optBest. O comprimento do tour em cada iteração é apresentado junto das figuras. A Figura 4.10, por sua vez, apresenta a iteração final obtida pelo algoritmo na iteração 29.



(a) Comprimento: 79,14 u.c. Iteração: 1.



(b) Comprimento: 67,37 u.c. Iteração: 10.



(c) Comprimento: 63,65 u.c. Iteração: 15.

(d) Comprimento: 62,96 u.c. Iteração: 20.

Figura 4.9: Evolução dos tours a cada iteração do método 2optBest.

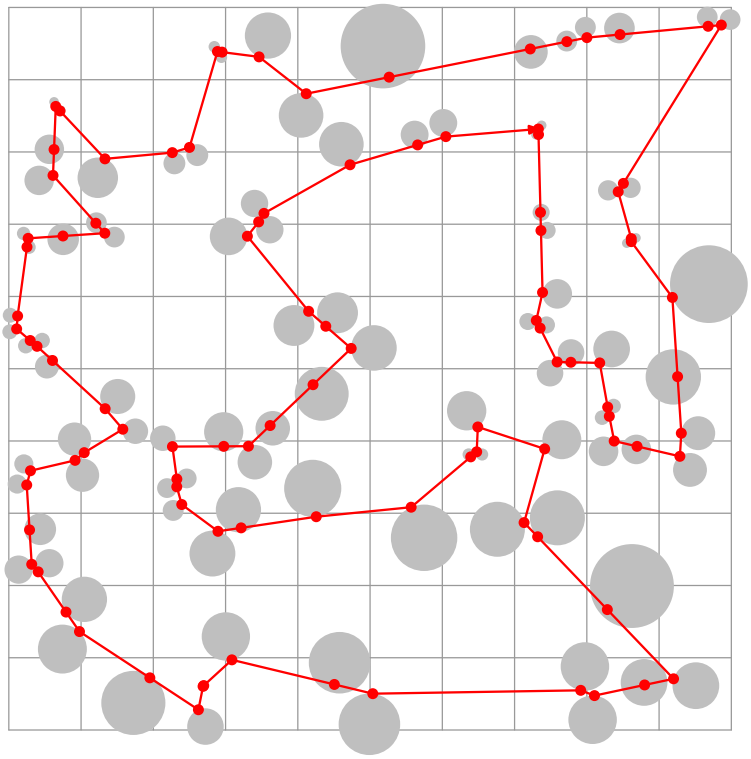
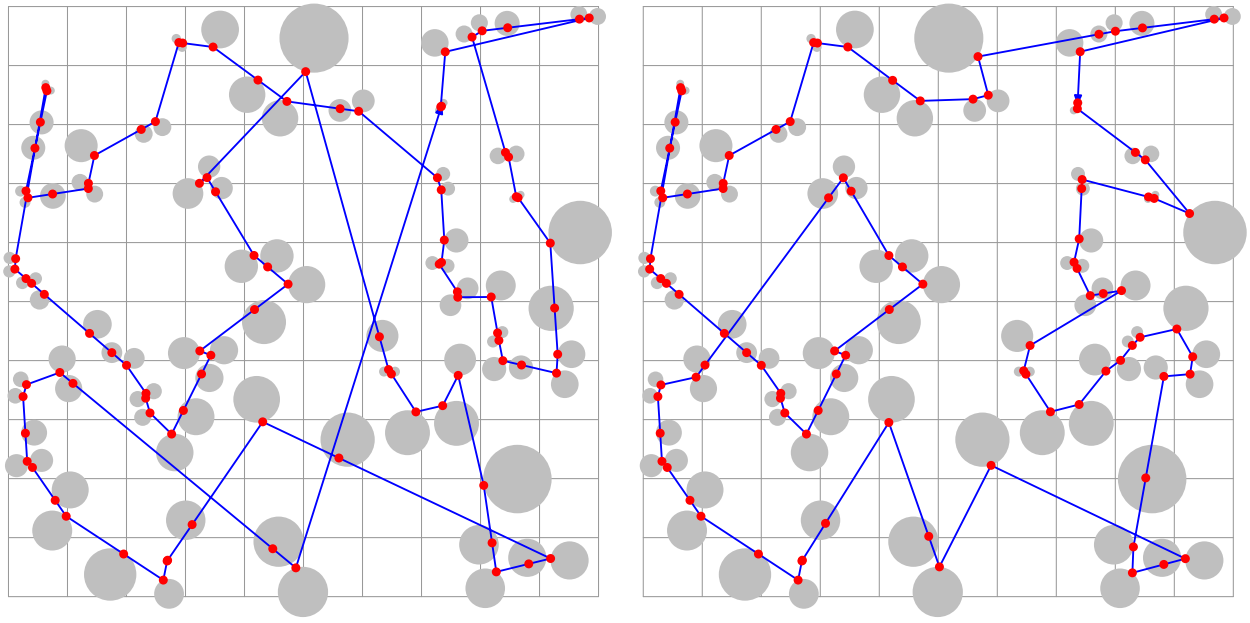


Figura 4.10: Solução encontrada pelo método 2optBest para a instância I2. Iteração: 29. Comprimento: 62,87 u.c. Tempo de execução: 17,6s.

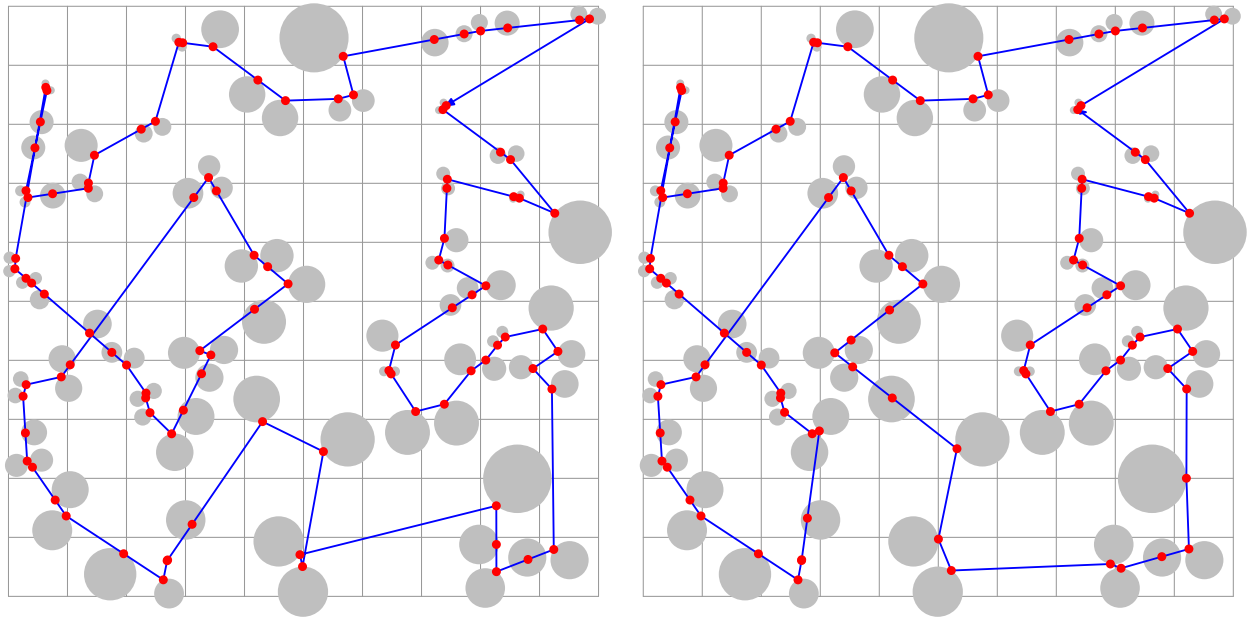
Já a Figura 4.11 exibe, respectivamente, as iterações 1, 20, 30 e 40 geradas com o Algoritmo 6 do método 2optFirst a partir do tour inicial construído na Figura 4.8. A solução encontrada por esse algoritmo, na iteração 66, é apresentada na Figura 4.12. A necessidade de mais iterações pelos métodos que utilizam a estratégia do *primeiro vizinho* que melhora é algo esperado, pois

cada iterando é atualizado assim que um tour de menor comprimento é encontrado. Com isso, em muitas iterações, essa "melhora" no comprimento do tour pode ser muito pequena, não resultando em ganhos significativos. Assim, apesar de ambos os Algoritmos 5 e 6 utilizarem movimentos do tipo 2opt para a construção da vizinhança, as soluções encontradas por eles serão diferentes em muitos casos, pois a escolha dos vizinhos a cada iteração ocorre de acordo com estratégias diferentes. Comparando as Figuras 4.9a e 4.11a por exemplo, observe que elas são distintas, tanto é que possuem comprimentos diferentes. Entretanto, ambas foram obtidas a partir de um movimento do tipo 2-Opt no tour inicial seguido de uma atualização dos pontos pelo método BCD (3).



(a) Comprimento: 81,62 u.c. Iteração: 1.

(b) Comprimento: 69,61 u.c. Iteração: 20.



(c) Comprimento: 68,21 u.c. Iteração: 30.

(d) Comprimento: 66,03 u.c. Iteração: 40.

Figura 4.11: Evolução dos tours a cada iteração do método 2optFirst.

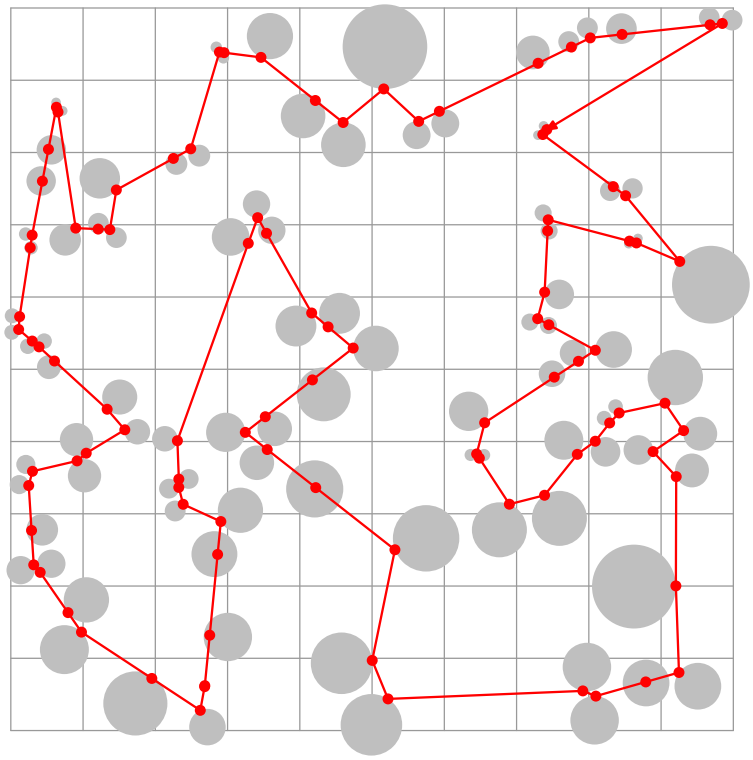
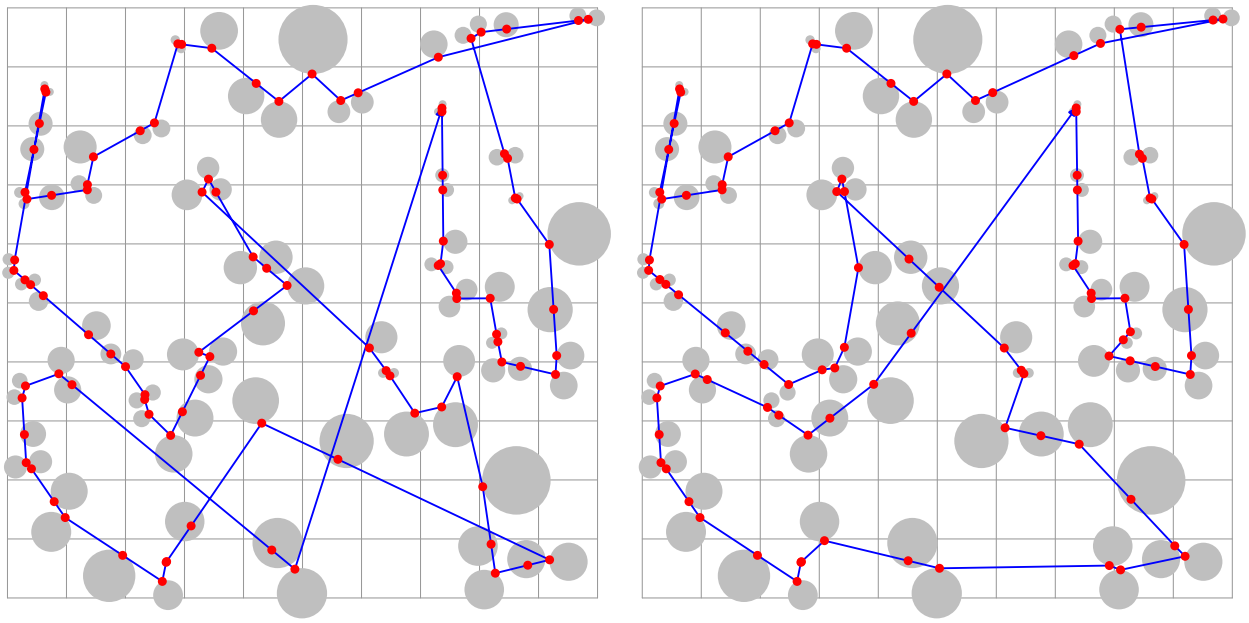


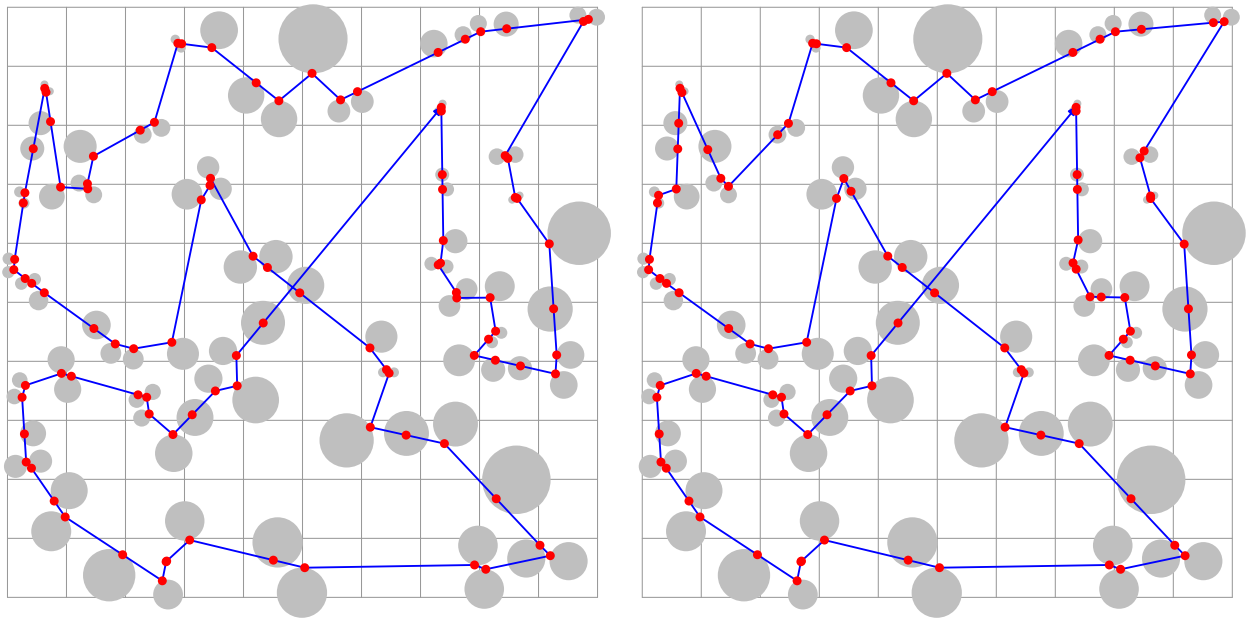
Figura 4.12: Solução encontrada pelo método 2optFirst para a instância I2. Iteração: 66. Comprimento: 64,26 u.c. Tempo de execução: 9,8s.

Na sequência, a Figura 4.13 apresenta as iterações 1, 20, 30 e 40 produzidas pelo Algoritmo 7 do método InsercaoBest. O critério de parada é atingido na iteração 42, com um tour cuja vizinhança não contém soluções melhores.



(a) Comprimento: 78,67 u.c. Iteração: 1.

(b) Comprimento: 69,61 u.c. Iteração: 20.



(c) Comprimento: 66,66 u.c. Iteração: 30.

(d) Comprimento: 66,20 u.c. Iteração: 40.

Figura 4.13: Evolução dos tours a cada iteração do método InsercaoBest.

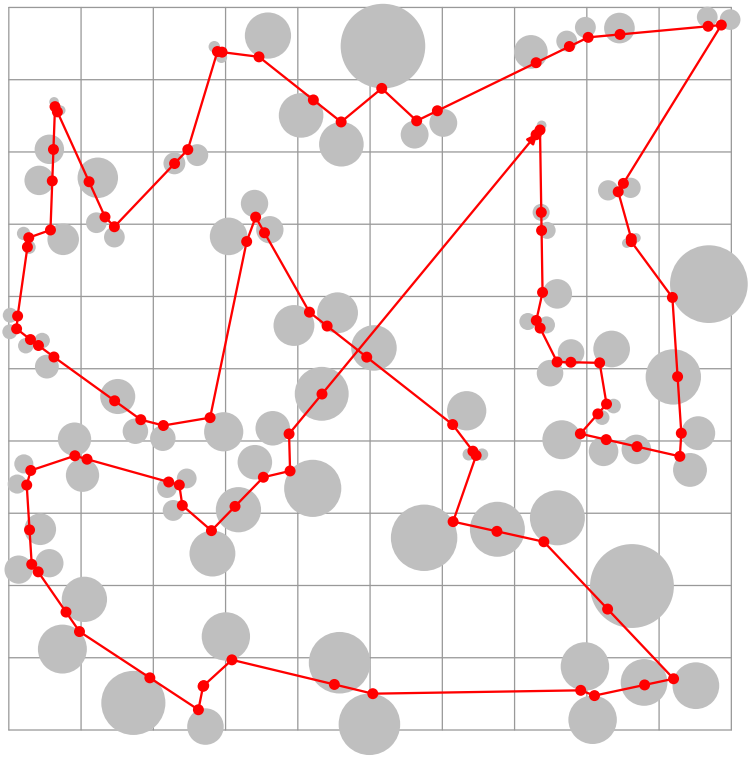
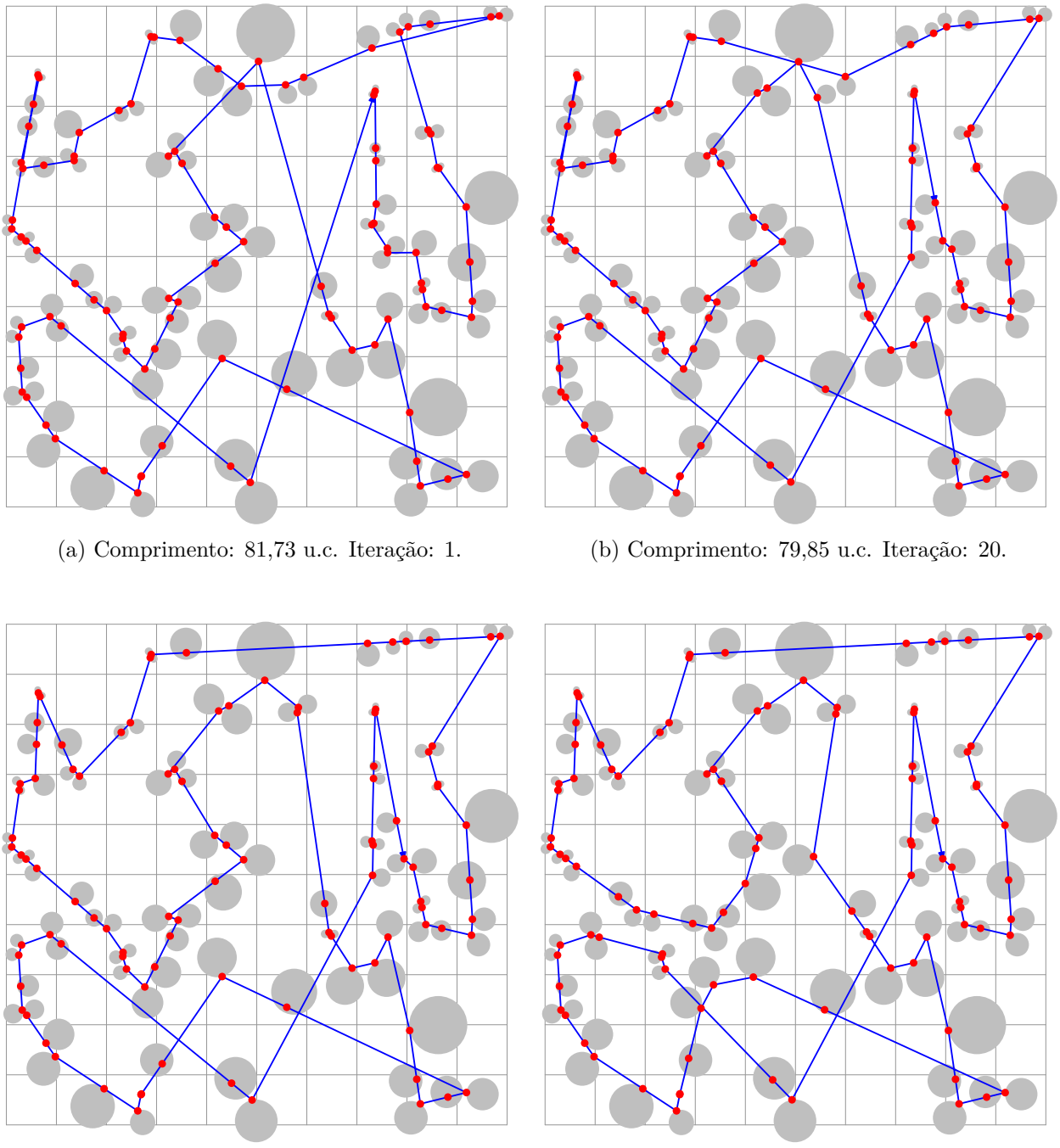


Figura 4.14: Solução encontrada pelo método InsercaoBest para a instância I2. Iteração: 42. Comprimento: 66,20 u.c. Tempo de execução: 52,42s.

E por fim, os iterandos do método InsercaoFirst do Algoritmo 8 são apresentados na Figura 4.15. Este algoritmo encontra uma solução na iteração 103.



(a) Comprimento: 81,73 u.c. Iteração: 1.

(b) Comprimento: 79,85 u.c. Iteração: 20.

(c) Comprimento: 78,58 u.c. Iteração: 30.

(d) Comprimento: 77,13 u.c. Iteração: 40.

Figura 4.15: Evolução dos tours a cada iteração do método InsercaoFirst.

Observe que para a Instância I2, o algoritmo InsercaoFirst (8) obteve a pior solução quando comparado aos demais, com comprimento de 67,29 u.c., enquanto que o algoritmo 2optBest (5) encontrou o tour de menor comprimento, 62,87 u.c. Analisando as figuras acima, podemos notar que os algoritmos de inserção não possuem a mesma tendência dos algoritmos 2opt de desfazer os segmentos que se cruzam. Este pode ser um dos motivos pelo quais os algoritmos de inserção costumam obter tours de maior comprimento na maioria das instâncias de 100 bolas. Para possibilitar conclusões mais contundentes acerca disso, as tabelas a seguir apresentam a performance

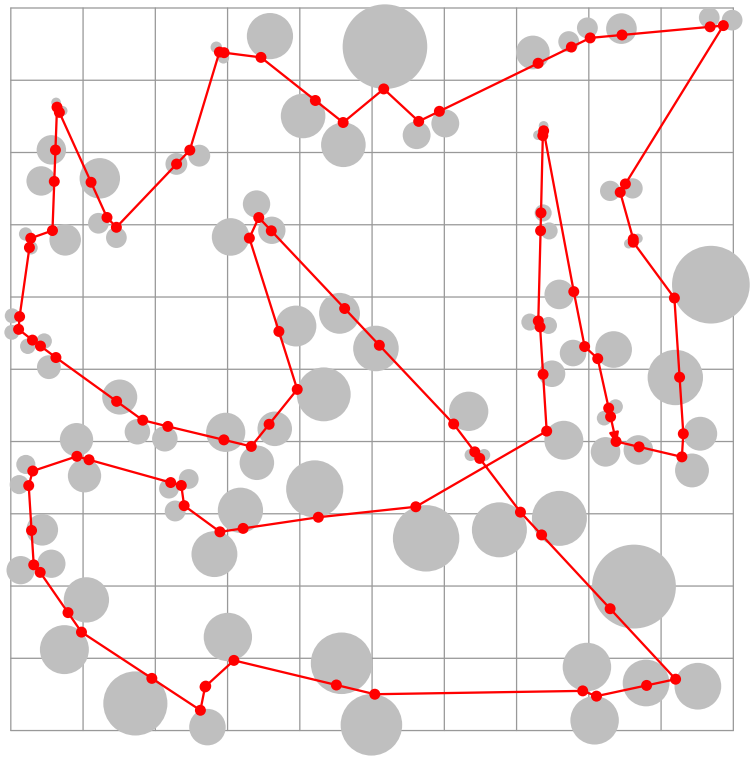


Figura 4.16: Solução encontrada pelo método InsercaoFirst para a instância I2. Iteração: 103. Comprimento: 67,29 u.c. Tempo de execução: 47,75s.

dos Algoritmos 5 a 8 quando aplicados a instâncias aleatórias com diferentes quantidades de bolas.

Experimentos numéricos também foram realizados para instâncias com $m = 50$, 200, 300, 400 e 500 bolas. As tabelas a seguir resumem os dados e resultados obtidos nestes testes, enquanto que as tabelas completas podem ser encontras no Apêndice. Para montagem destas tabelas, cada método foi testado em 10 instâncias aleatórias para cada valor de m acima, assim como feito para $m = 100$. Em seguida, para cada método em separado e cada valor de m, calculamos a média aritmética entre os respectivos dados obtidos nas 10 instâncias. Com estes valores, montamos as tabelas a seguir. As Tabelas 4.2 e 4.3 apresentam os dados gerados com os métodos 2optBest (5) e 2optFirst (6), respectivamente. Os dados referentes aos métodos InsercaoBest (7) e InsercaoFirst (8) são exibidos nas tabelas Tabelas 4.4 e 4.5.

m	#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
			#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
50	11,8	45,2184513877	13.876,8	95.126,6	1.176,0	8.127,1	6,9	35,94	14,06	1,07
100	23,4	61,9056970947	113.513,4	743.248,8	4.851,0	31.760,2	6,5	71,39	28,61	14,18
200	40,8	103,9599841500	803.800,8	5.112.433,4	19.701,0	125.265,5	6,4	139,39	60,61	179,12
300	59,0	126,5987904512	2.628.509,0	16.341.632,0	44.551,0	277.501,6	6,2	206,79	93,21	835,68
400	77,0	169,2011441248	6.113.877,0	38.308.193,6	79.401,0	497.637,7	6,3	274,60	125,40	2.581,46
500	95,6	186,9254532602	11.878.395,6	73.127.485,2	124.251,0	765.360,5	6,2	346,54	153,46	6.140,89

Tabela 4.2: Desempenho do método 2optBest (5) em instâncias aleatórias.

m	#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
			#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
50	35,6	45,9188512433	9.989,4	66.581,6	307,0	2.042,8	6,6	35,36	14,64	0,78
100	89,6	63,2111098248	126.339,4	836.215,4	1.419,3	9.402,4	6,6	70,14	29,86	15,78
200	190,4	105,2398390218	952.127,1	6.017.527,3	5.472,3	34.570,0	6,3	137,07	62,93	209,48
300	243,7	129,6899504207	3.066.375,4	19.182.424,8	12.877,2	80.910,7	6,2	201,59	98,41	973,10
400	376,4	172,7434158161	7.315.047,2	45.739.319,7	19.940,7	124.497,5	6,3	272,07	127,93	3.073,22
500	481,9	190,5349783970	13.820.841,2	84.910.925,6	29.041,3	178.285,4	6,2	338,42	161,58	7.066,65

Tabela 4.3: Desempenho do método 2optFirst (6) em instâncias aleatórias.

m	#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
			#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
50	16,3	47,0093060724	39.935,0	274.666,8	2.450,0	16.828,9	6,9	35,93	14,07	2,87
100	36,9	67,9608319122	365.310,0	2.482.412,2	9.900,0	67.068,5	6,8	70,04	29,96	44,84
200	63,9	114,5187360983	2.543.220,0	17.059.952,7	39.800,0	266.722,6	6,7	137,64	62,36	580,79
300	84,4	139,2892076064	7.570.680,0	50.627.048,7	89.700,0	598.196,0	6,7	204,22	95,78	2.538,64
400	119,9	186,1277456264	19.136.040,0	128.624.477,1	159.600,0	1.074.067,5	6,7	274,40	125,60	8.557,06
500	146,0	205,8914173523	36.427.000,0	242.114.775,4	249.500,0	1.658.276,6	6,6	342,44	157,56	20.024,07

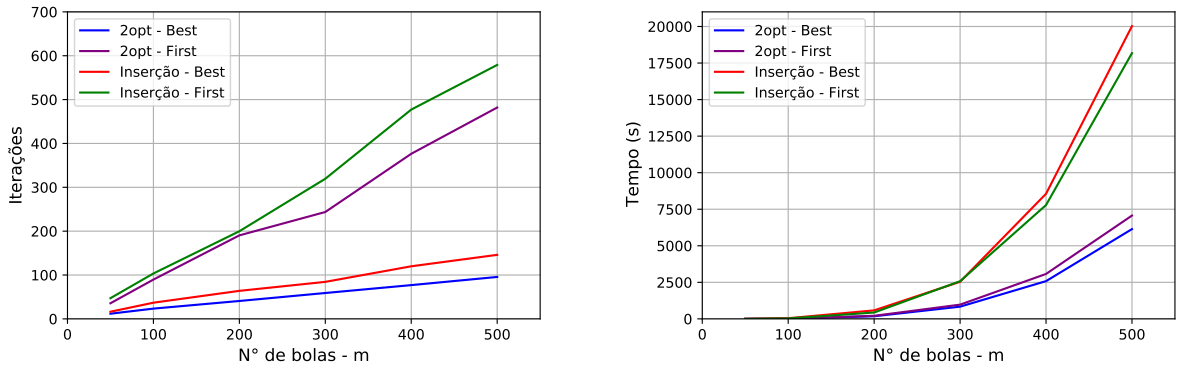
Tabela 4.4: Desempenho do método InsercaoBest (7) em instâncias aleatórias.

m	#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
			#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
50	47,2	48,8951847818	25.226,8	173.636,3	617,7	4.264,2	6,9	35,03	14,97	1,87
100	103,5	67,9047963421	224.623,7	1.562.096,7	2.295,1	16.083,7	6,9	68,65	31,35	28,32
200	199,8	115,8363532381	1.901.680,6	12.780.899,8	9.833,5	66.159,8	6,7	134,50	65,50	435,16
300	319,5	139,8506718107	7.670.746,6	51.458.245,1	24.488,8	164.259,3	6,7	202,57	97,43	2.575,41
400	477,2	188,4332277022	17.330.054,5	117.580.035,3	36.670,5	248.804,1	6,8	268,63	131,37	7.785,33
500	579,0	207,8850985607	32.921.975,7	220.437.173,2	58.892,1	393.555,3	6,7	338,39	161,61	18.178,03

Tabela 4.5: Desempenho do método InsercaoFirst (8) em instâncias aleatórias.

Uma análise inicial das tabelas acima nos permite observar que, no caso das instâncias aleatórias, independente da quantidade de bolas, o método BCD (3) mantém uma certa estabilidade na média de ciclos que ele realiza a cada chamada, entre 6,2 e 6,9 ciclos. Além disso, na atualização dos pontos a cada ciclo, entre 67% e 70% deles são computados pelo método de Newton, indicando que no caso de instâncias aleatórias a maioria dos pontos tende a estar localizada na fronteira das bolas.

Em relação ao tempo de execução e número de iterações, os gráficos a seguir, construídos a partir das tabelas, nos auxiliam nas conclusões.



(a) Iterações em função do número de bolas.

(b) Tempo em função do número de bolas.

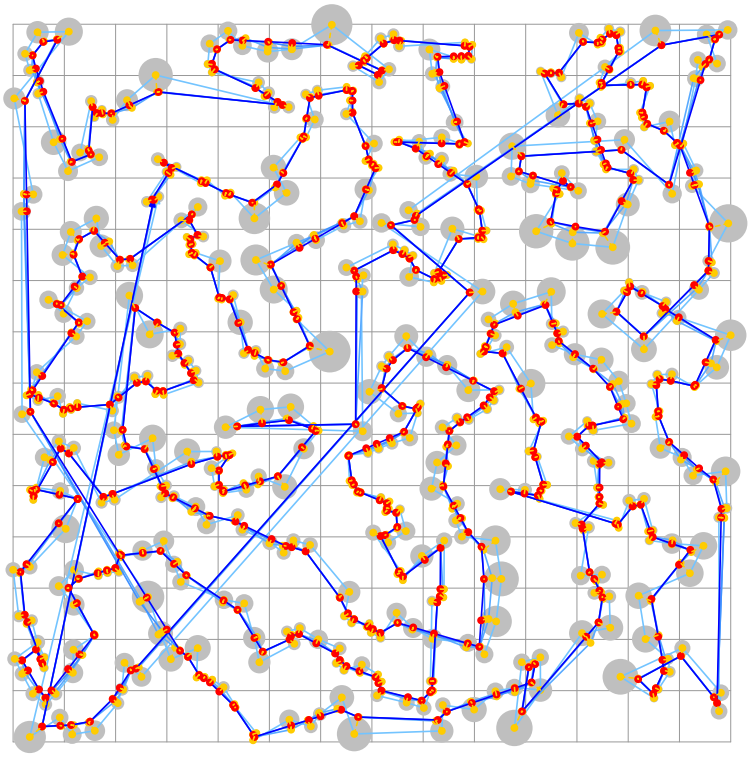
Figura 4.17: Gráficos das iterações e do tempo, respectivamente, em função do número de bolas.

Assim como observado nos testes com as instâncias de 100 bolas, o método 2optBest (5) consegue obter, em média, soluções de melhor qualidade, i.e., menor comprimento, e em menos iterações. Este método também encontra estas soluções mais rapidamente que os demais métodos testados. Apesar de o número de iterações em função do número de bolas apresentar um crescimento linear para todos os métodos, com mudanças na inclinação que é maior com os métodos de inserção, o mesmo não é observado em relação ao crescimento do tempo de execução em função do número de bolas, o qual apresenta um comportamento quadrático.

Como já observado nos tours gerados para as instâncias de 100 bolas, uma das razões de os métodos 2-Opt obterem melhores soluções está no fato destes métodos desfazerem as arestas que geram sobreposições na rota do tour, característica que não é observada nos métodos de inserção. Esse comportamento também pode ser visualizado nas soluções encontradas para a instância I9 de 500 bolas. O processo de construção do tuor inicial para a instância I9 é apresentado na Figura 4.18 e as soluções podem ser visualizadas na Figura 4.19.

Para a instância I9 o método BCD levou cerca de 25 ciclos para concluir a construção do tour inicial, que está representado na cor azul com os pontos em vermelho na Figura 4.18. Com apenas um ciclo do BCDM o comprimento do tour decaiu de 285,79 u.c. para 233,85 u.c., uma diferença de 51,85 u.c. Entre os ciclos 3 e 25 apenas pequenas alterações na localização dos pontos foram realizadas, não impactando significativamente no custo total do tour.

Em relação às soluções encontradas por cada algoritmo, temos que os algoritmos 2optBest (5) e 2optFirst (6) encontraram soluções distintas (ver Figuras 4.19a e 4.19b) mas com custo muito próximo, 183,25 u.c. e 183,45 u.c. respectivamente. No entanto, enquanto o algoritmo 2optBest encontrou uma solução em 92 iterações, para o algoritmo 2optFirst foram necessárias 457 iterações. Mesmo assim, o tempo de execução do 2optFirst foi menor. Isso ocorre porque este algoritmo possui iterações mais baratas devido a estratégia do *primeiro vizinho* que melhora e, portanto, o 2optFirst avalia a mesma quantidade de vizinhos que uma iteração do algoritmo 2optBest somente no pior

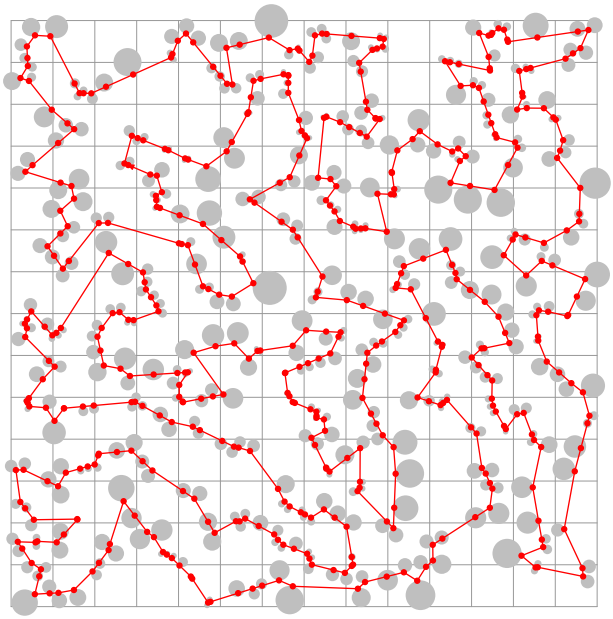


Ciclos	Comprimento do tour
0	285,7096
1	233,8532
2	232,2656
3	232,1103
4	232,0864
5	232,0805
6	232,0792
7	232,0788
8	232,0786
⋮	⋮
25	232,0785

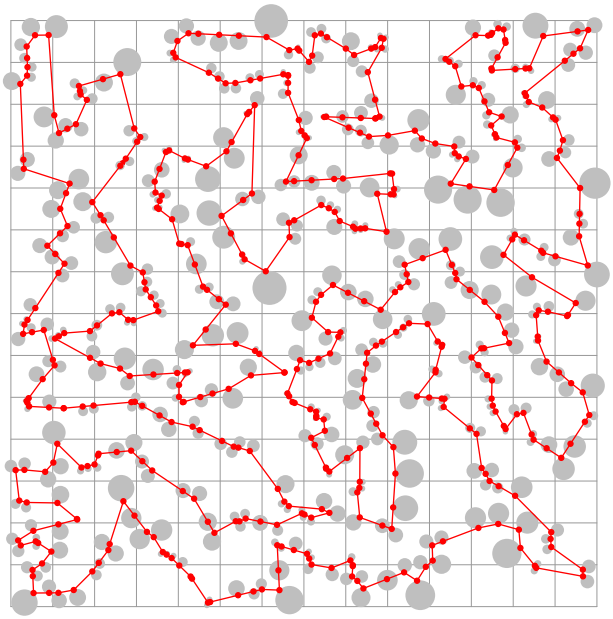
Figura 4.18: Construção do tour inicial para a instância I9 de 500 bolas.

caso. Com efeito, enquanto que o algoritmo 2optBest avaliou cerca de 124.251,0 vizinhos por iteração, o algoritmo 2optFirst avaliou, em média, apenas 24.702,6 vizinhos por iteração.

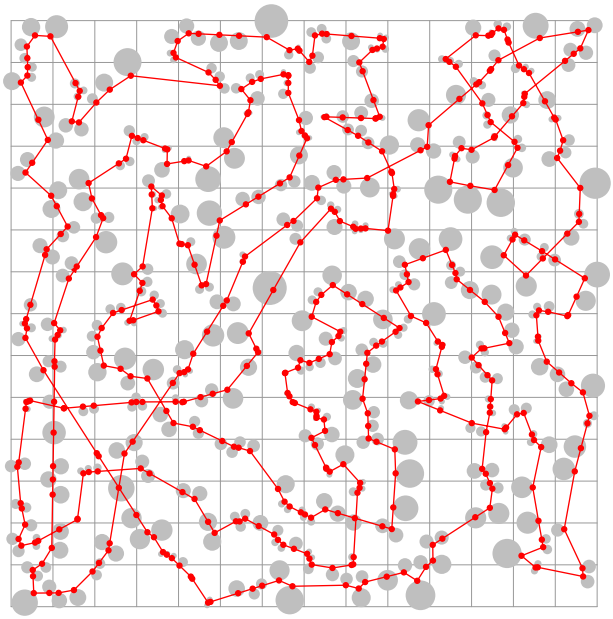
O algoritmo InsercaoBest (7), por sua vez, demandou cerca de quatro vezes mais tempo de execução que o algoritmo 2optBest, apesar de não encontrar uma solução tão boa quanto. Essa diferença de tempo se deve a dois fatores. Primeiro, o número de vizinhos avaliados a cada iteração do InsercaoBest é duas vezes maior. E segundo, ele levou mais iterações até atingir algum critério de parada. Agora, quando comparamos InsercaoBest com o algoritmo InsercaoFirst (8), as mesmas observações feitas quando comparamos os algoritmos 2optBest e 2optFirst se aplicam. O algoritmo InsercaoFirst possui iterações mais baratas que o InsercaoBest e por esse motivo seu tempo de execução foi menor, apesar de levar mais iterações até parar, um total de 456 iterações, enquanto que o InsercaoBest necessitou de 159 iterações.



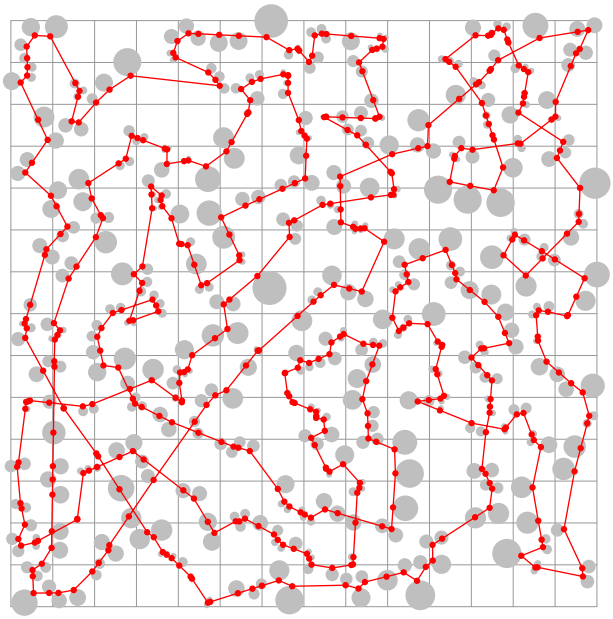
(a) Tour ótimo com 2optBest: 183,25 u.c. Iteração: 92. Tempo de execução: 5.862,31s.



(b) Tour ótimo com 2optFirst: 183,45 u.c. Iteração: 457. Tempo de execução: 5.699,38s.



(c) Tour ótimo com InsercaoBest: 202,93 u.c. Iteração: 159. Tempo de execução: 21.823,84s.



(d) Tour ótimo com InsercaoFirst: 202,76 u.c. Iteração: 456. Tempo de execução: 17.228,55s.

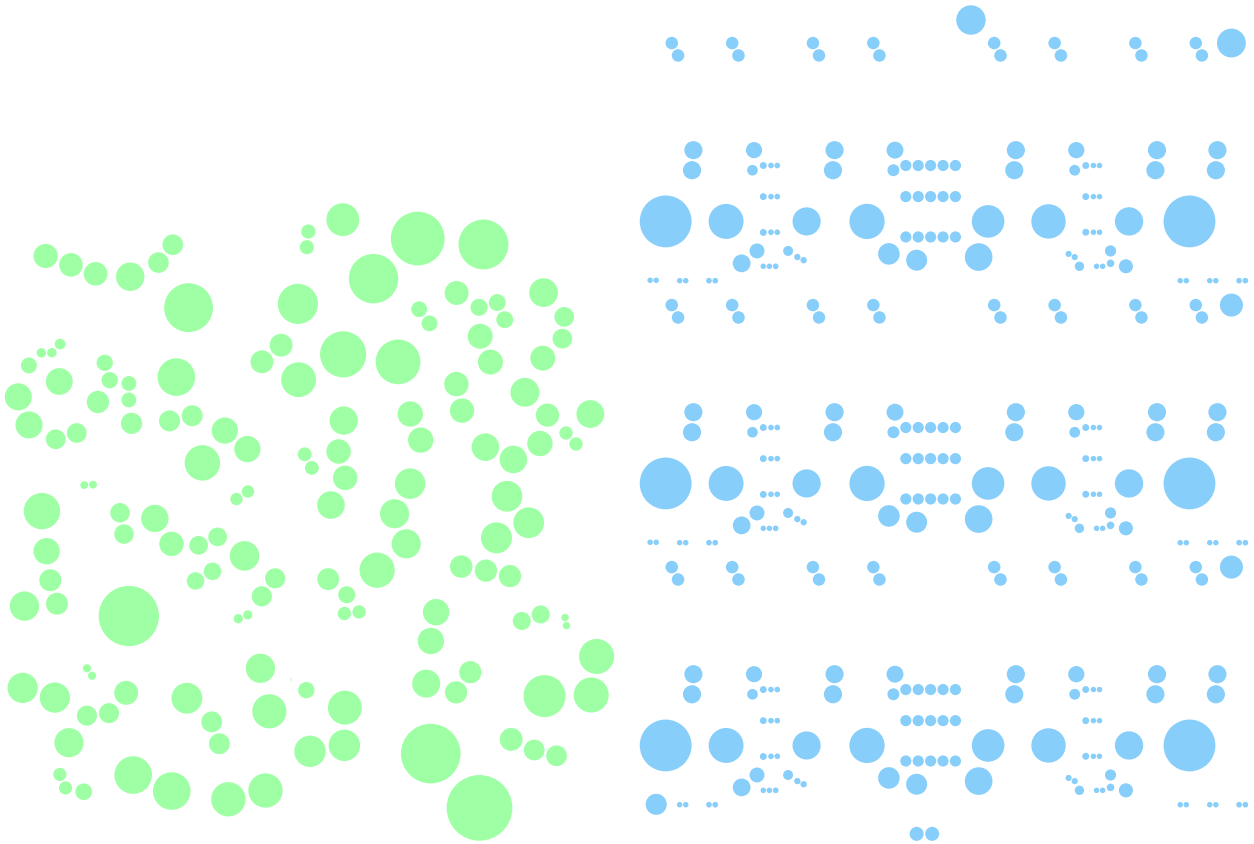
Figura 4.19: Soluções encontradas por cada um dos métodos para a instância I9 com 500 bolas.

4.3 Experimentos com instâncias da biblioteca TSPLIB

Nesta seção detalhamos os experimentos numéricos realizados com instâncias da biblioteca TSPLIB [28] e avaliamos o desempenho dos algoritmos para este conjunto de dados. A biblioteca TSPLIB é um repositório de instâncias para o TSP e para algumas variantes suas. Para os experimentos, selecionamos 10 instâncias Euclidianas do $\mathbb{R}^2$. Tais instâncias são compostas por

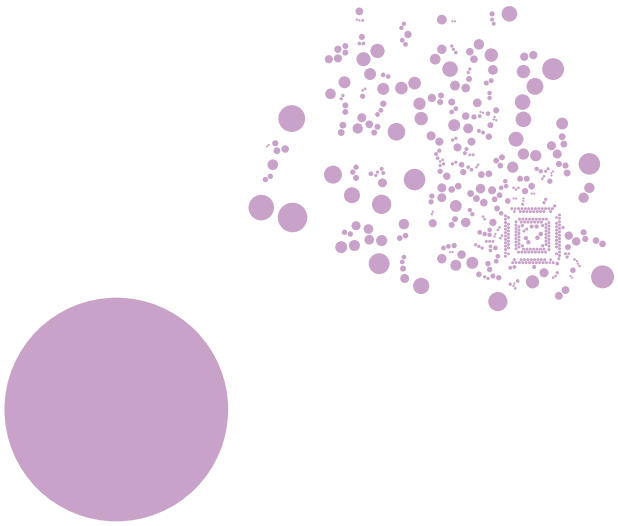
coordenadas de pontos do $\mathbb{R}^2$ e para transformá-las em instâncias do CTSP, consideramos os pontos fornecidos como sendo os centros das bolas $\mathcal{B}_i$ e definimos os raios utilizando a estratégia (4.1) da seção anterior com $\alpha = 0{,}9$.

A Figura 4.20 apresenta algumas das instâncias selecionadas para os testes.



(a) Instância *ch150*.

(b) Instância *lin318*.



(c) Instância *d493*.

(d) Instância *tsp225*.

Figura 4.20: Exemplos de instâncias da biblioteca TSPLIB adaptadas ao CTSP.

O principal critério utilizado na seleção das instâncias da TSPLIB foi o número de pontos, optando por problemas de tamanhos variados e que, em sua maioria, não ultrapassassem a quan-

tidade de 500 pontos. Isso porque nos testes com 500 bolas aleatórias já foi possível constatar a necessidade de um tempo de processamento considerável. Além disso, problemas cujo número de bolas fosse próximo do tamanho das instâncias aleatórias era de nosso interesse para possibilitar comparações entre o desempenho dos algoritmos com as instâncias aleatórias e o desempenho obtido com as instâncias da TSPLIB.

A nomenclatura dessas instâncias sugere a quantidade de pontos que às compõem. Assim, a instância *ch150* é composta por 150 pontos e a instância *d493* por 493 pontos, por exemplo. Os autores e suas motivações por detrás da criação das instâncias da biblioteca TSPLIB são diversos. As coordenadas da instância *berlin52*, por exemplo, correspondem à 52 localidades em Berlim, enquanto que os pontos de *bier127* se referem à 127 *Biergaerten* da cidade de Augsburg, na Alemanha. As instâncias *d493* e *p654*, por sua vez, estão relacionadas ao problema de perfuração de placas de circuitos (*drilling problem* [2, p. 69]).

Vejamos agora o desempenho dos algoritmos do Capítulo 3 com as instâncias da TSPLIB. As Tabelas 4.6 e 4.7 correspondem aos resultados obtidos pelos algoritmos 2optBest (5) e 2optFirt (6), respectivamente, enquanto que o desempenho dos algoritmos InsercaoBest (7) e InsercaoFirst (8) é apresentado nas Tabelas 4.8 e 4.9. Nas tabelas, a primeira coluna corresponde ao nome da instância, o qual sugere a quantidade de bolas. A coluna seguinte apresenta o número de iterações que o algoritmo levou e a terceira coluna contém o comprimento da solução. As colunas 4 e 5 correspondem, respectivamente, ao total de chamadas e ciclos do método BCD (5) realizados pelo algoritmo até convergir, e as colunas 6 e 7 contêm a quantidade de chamadas e ciclos do BCDM por iteração do algoritmo. A coluna 8 apresenta a média de ciclos que o método BCD realiza quando é chamado, e as colunas 9 e 10 indicam, em média, quantos pontos são calculados pelo método de Newton (2) e quantos pela estratégia da projeção (1) a cada ciclo. Por fim, a última coluna exibe o tempo de execução do algoritmo.

Instância	#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
			#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
berlin52	17	5.633,25	21.675	174.029	1.275,0	10.237,0	8,0	38,70	13,30	2,13
eil101	27	443,56	133.650	1.078.954	4.950,0	39.961,3	8,1	72,55	28,45	21,23
bier127	34	90.443,14	267.750	2.413.862	7.875,0	70.995,9	9,0	95,59	31,41	59,33
ch150	21	4.986,02	231.546	1.955.277	11.026,0	93.108,4	8,4	101,94	48,06	56,21
rat195	26	1.838,76	486.746	5.542.144	18.721,0	213.159,4	11,4	111,37	83,63	186,51
tsp225	34	3.168,21	849.184	7.902.126	24.976,0	232.415,5	9,3	141,99	83,01	313,82
pr264	36	45.340,69	1.240.308	30.198.758	34.453,0	838.854,4	24,3	98,61	165,39	1.160,43
lin318	66	36.575,50	3.305.676	37.869.292	50.086,0	573.777,2	11,5	206,13	111,87	2.113,27
d493	112	26.885,68	13.528.032	188.500.778	120.786,0	1.683.042,7	13,9	297,06	195,94	15.518,41
p654	221	28.079,00	47.046.038	455.281.706	212.878,0	2.060.098,2	9,7	443,46	210,54	49.338,99

Tabela 4.6: Desempenho do método 2optBest (5) em instâncias da biblioteca TSPLIB.

Instância	#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
			#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
berlin52	34	5.647,92	12.884	111.934	378,9	3.292,2	8,7	35,94	16,06	1,39
eil101	123	480,22	126.199	1.000.470	1.026,0	8.133,9	7,9	69,44	31,56	19,47
bier127	188	91.371,51	353.291	3.318.893	1.879,2	17.653,7	9,4	89,59	37,41	80,03
ch150	75	5.203,94	229.157	1.919.060	3.055,4	25.587,5	8,4	97,44	52,56	53,41
rat195	112	1.869,77	698.389	7.324.478	6.235,6	65.397,1	10,5	113,91	81,09	247,31
tsp225	340	2.953,02	1.595.585	13.899.776	4.692,9	40.881,7	8,7	148,54	76,46	569,13
pr264	172	44.395,41	1.589.029	38.733.775	9.238,5	225.196,4	24,4	95,50	168,50	1.485,85
lin318	305	37.349,23	3.624.471	44.529.556	11.883,5	145.998,5	12,3	196,89	121,11	2.427,65
d493	733	27.291,22	17.905.318	226.338.641	24.427,4	308.784,0	12,6	283,80	209,20	18.459,19
p654	1207	28.390,45	107.625.705	2.144.456.571	89.167,9	1.776.683,2	19.9	417,40	236,60	219.778,68

Tabela 4.7: Desempenho do método 2optFirst (6) em instâncias da biblioteca TSPLIB.

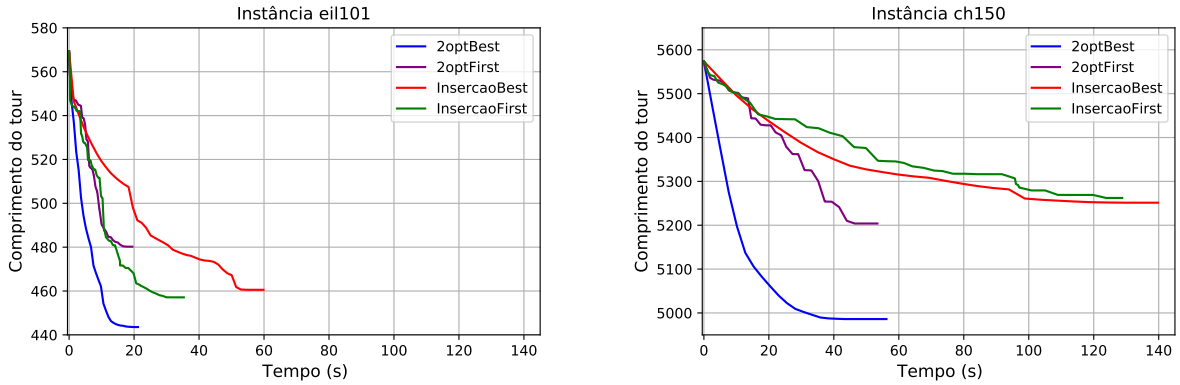
Instância	#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
			#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
berlin52	11	6.193,36	29.172	252.531	2.652,0	22.957,4	8,7	38,34	13,66	2,93
eil101	42	460,55	424.200	3.135.832	10.100,0	74.662,7	7,4	75,36	25,64	59,83
bier127	33	99.452,48	528.066	4.884.423	16.002,0	148.012,8	9,2	92,36	34,64	114,83
ch150	27	5.251,36	603.450	5.053.974	22.350,0	187.184,2	8,4	105,44	44,56	139,80
rat195	36	2.036,89	1.361.880	14.536.672	37.830,0	403.796,4	10,7	112,19	82,81	483,46
tsp225	63	3.225,95	3.175.200	30.843.096	50.400,0	489.573,0	9,7	145,91	79,09	1.217,16
pr264	58	47.705,79	4.027.056	89.716.895	69.432,0	1.546.843,0	22,3	113,25	150,75	3.416,85
lin318	114	39.888,83	11.491.884	132.746.006	100.806,0	1.164.438,6	11,6	202,94	115,06	7.163,22
d493	216	28.020,69	52.392.096	676.143.826	242.556,0	3.130.295,5	12,9	291,50	201,50	53.637,98
p654	290	31.176,85	123.847.980	2.297.178.518	427.062,0	7.894.084,25	18,55	388,80	265,21	227.610,05

Tabela 4.8: Desempenho do método InsercaoBest (7) em instâncias da biblioteca TSPLIB.

Instância	#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
			#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
berlin52	58	6.262,70	56.264	498.407	970,1	8.593,2	8,9	36,71	15,29	5,45
eil101	116	457,11	245.266	1.782.938	2.114,4	15.370,2	7,3	76,43	24,57	35,32
bier127	180	100.744,58	569.803	4.630.808	3.165,6	25.726,7	8,1	98,22	28,78	111,19
ch150	79	5.262,21	549.683	4.535.461	6.958,0	57.410,9	8,3	108,28	41,72	128,77
rat195	110	2.031,47	1.510.051	16.182.274	13.727,7	147.111,6	10,7	112,89	82,11	536,25
tsp225	186	3.495,42	2.445.364	23.064.284	13.147,1	124.001,5	9,4	142,38	82,62	917,46
pr264	149	47.409,40	3.966.536	90.369.178	26.621,0	606.504,6	22,8	106,55	157,45	3.449,93
lin318	487	40.480,53	12.317.857	136.231.896	25.293,3	279.737,0	11,1	194,67	123,33	7.279,02
d493	947	28.805,17	82.067.839	824.625.896	86.660,9	870.777,1	10,0	314,12	178,88	68.215,80
p654	993	30.137,85	156.716.307	2.670.213.890	157,662.28	2.686.331,9	17,03	397,63	256,37	269.301,09

Tabela 4.9: Desempenho do método InsercaoFirst (8) em instâncias da biblioteca TSPLIB.

Inicialmente, analisando as tabelas acima, é possível observar que o número de iterações não necessariamente aumenta com o número de bolas. Por exemplo, todos os quatro algoritmos obtêm uma solução para a instância *ch150* em menos iterações do que para as instâncias *eil101* e *bier127*. Com efeito, o algoritmo 2optFirst, por exemplo, leva 75 iterações para encontrar uma solução para a instância *ch150* e 123 iterações para a instância *eil101*. Apesar disso, quando olhamos para o tempo de execução (veja Figura 4.21), este ainda é maior para a instância de 150 bolas.



(a) *eil101*. (b) *ch150*.

Figura 4.21: Gráficos do comprimento do tour em função do tempo.

O crescimento do tempo de execução em função do número de bolas m é algo esperado, pois as heurísticas de busca local que compõem os algoritmos são $O(m^2)$. Assim, mesmo que o algoritmo necessite menos iterações para convergir, o número de vizinhos avaliados a cada iteração é maior quanto maior for o número de bolas, tornando as iterações mais custosas. Além disso, os ciclos realizados pelo método BCD também são mais caros, pois o número de pontos atualizados a cada ciclo é maior.

Na grande maioria dos testes realizados o algoritmo 2optBest (5) encontrou soluções melhores, isto é, de menor comprimento, que os demais métodos. Este é o caso para a instância *eil101*. No entanto, diferentemente do que é observado para as demais instâncias, para *eil101* os Algoritmos 7 e 8, InsercaoBest e InsercaoFirst, encontraram soluções melhores que o algoritmo 2optFirst (6).

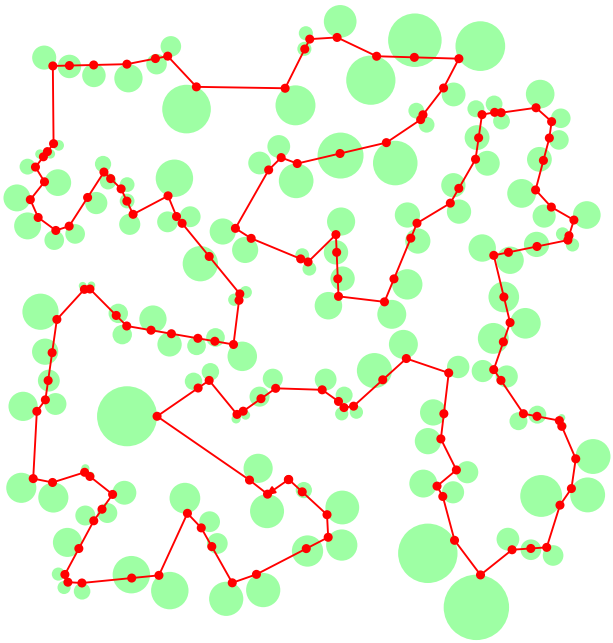
Para muitas instâncias da biblioteca TSPLIB [28] as soluções ótimas para o problema do caixeiro viajante clássico são conhecidas, e também podem ser encontradas no repositório. Como no CTSP o tour não precisa passar exatamente pelos pontos fornecidos nas instâncias, pode-se esperar que o custo das soluções encontradas para o CTSP seja menor que o custo das soluções do TSP. A Tabela 4.10 confirma esta ideia. Nesta tabela, a primeira coluna contém o nome de cada instância e as demais colunas apresentam o comprimento das soluções ótimas para o TSP clássico (coluna 2) e das soluções do CTSP encontradas pelo nossos algoritmos.

Instância	TSP clássico	CTSP			
	Solução ótima	2optBest	2optFirst	InsercaoBest	InsercaoFirst
berlin52	7.542,00	5.633,25	5.647,92	6.193,36	6.262,70
eil101	629,00	443,56	480,22	460,55	457,11
bier127	118.282,00	90.443,14	91.371,51	99.452,48	100.744,58
ch150	6.528,00	4.986,02	5.203,94	5.251,36	5.262,21
rat195	2.323,00	1.838,76	1.869,77	2.036,89	2.031,47
tsp225	3.919,00	3.168,21	2.953,02	3.225,95	3.495,42
pr264	49.135,00	45.340,69	44.395,41	47.705,79	47.409,40
lin318	42.029,00	36.575,50	37.349,23	39.888,83	40.480,53
d493	35.002,00	26.885,68	27.291,22	28.020,69	28.805,17
p654	34.643,00	28.079,00	28.390,45	31.176,85	30.137,85

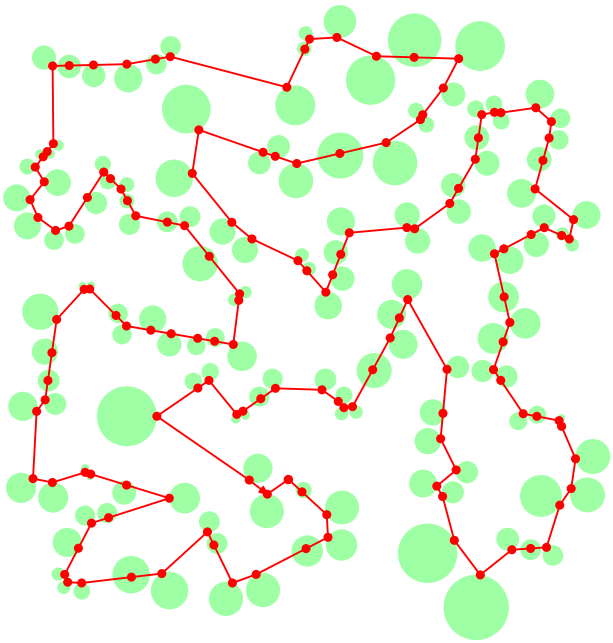
Tabela 4.10: Comprimento da solução ótima conhecida para cada instância considerando o TSP clássico e das soluções encontradas pelos algoritmos para o CTSP.

A seguir, apresentamos as figuras das soluções encontradas por cada um dos quatro algoritmos para as instâncias *ch150*, *pr264* e *d493*.

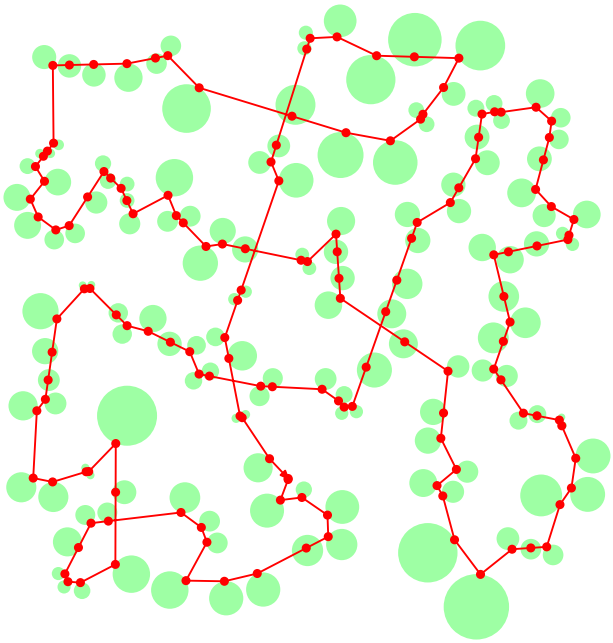
A Figura 4.22 contém as soluções para a instância *ch150*. Para este conjunto de dados o tour de menor comprimento, com 4.986,02 u.c, foi encontrado pelo algoritmo 2optBest em 21 iterações, enquanto que a pior solução, com um comprimento de 5.262,21 u.c., foi obtida pelo algoritmo InsercaoFirst em 79 iterações. Assim como nos experimentos com instâncias aleatórias, neste exemplo é possível observar que os métodos de inserção não conseguem desfazer os caminhos que se cruzam e por isso geram tours de comprimento maior.



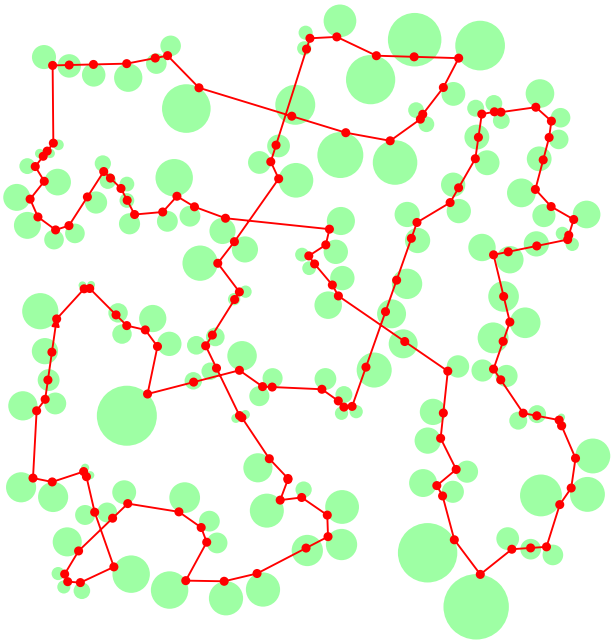
(a) Solução com 2optBest: 4.986,02 u.c. Iteração: 21. Tempo de execução: 56,21s.



(b) Solução com 2optFirst: 5.203,94 u.c. Iteração: 75. Tempo de execução: 53,41s.



(c) Solução com InsercaoBest: 5.251,36 u.c. Iteração: 27. Tempo de execução: 139,80s.



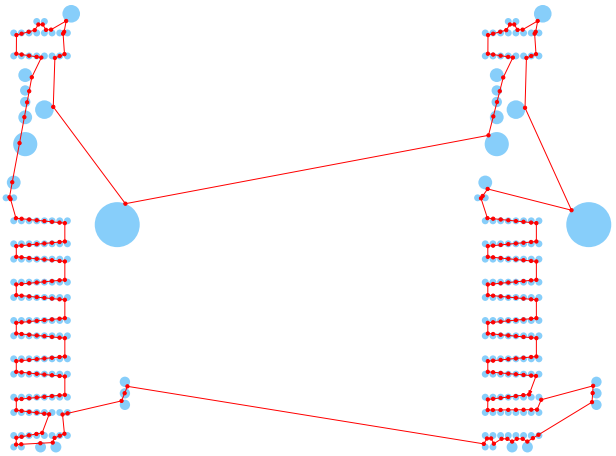
(d) Solução com InsercaoFirst: 5.262,21 u.c. Iteração: 79. Tempo de execução: 128,77s.

Figura 4.22: Soluções encontradas por cada um dos métodos para a instância *ch150*.

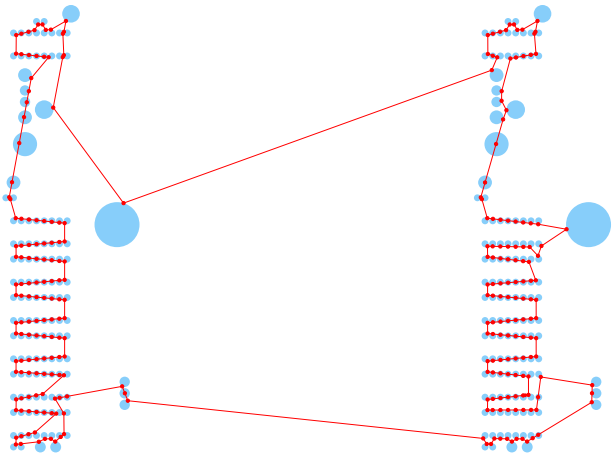
Na Figura 4.23 apresentamos as soluções encontradas para a instância *pr264*. Para esta instância o tour de menor comprimento é encontrado pelo algoritmo 2optFirst (6), com 44.395,41 u.c. em 172 iterações. É também com esta instância que a média de ciclos do método BCD por chamada atinge seu maior valor, ficando entre 22,3 e 24,4 a depender do algoritmo. Mesmo instâncias maiores, como *d493* e *p654*, não atingem essa média, indicando que a média de ciclos por chamada do

BCDM pode estar atrelada às características e complexidade da instância e não necessariamente ao número de bolas. Tanto é que para as instâncias aleatórias a média se manteve uniforme, mesmo variando o número de bolas.

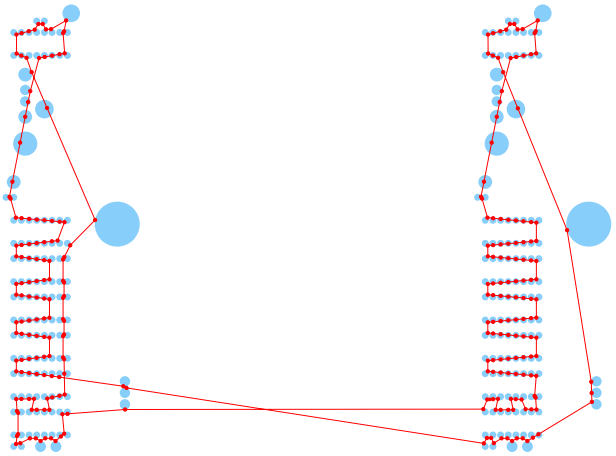
Ademais, entre todas as instâncias testadas nos experimentos numéricos, a instância *pr264* foi a única para a qual observamos que a maior parte dos pontos é ajustada com a estratégia da projeção (1). Nos demais casos, o método de Newton era o mais utilizado. Mas o motivo fica evidente quando analisamos as soluções na Figura 4.23, pois nesta instância muitas bolas estão posicionadas uma ao lado das outras, de modo que com uma ordenação adequada caímos no contexto em que o segmento que liga os pontos $x^{i_{\nu-1}}$ e $x^{i_{\nu+1}}$ de um tour intersecta a bola $\mathcal{B}_{i_\nu}$ em mais de um ponto.



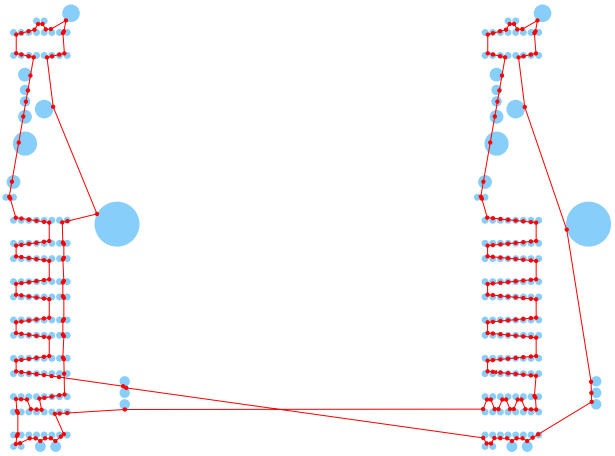
(a) Solução com 2optBest: 45.340,69 u.c. Iteração: 36. Tempo de execução: 1.160,43s.



(b) Solução com 2optFirst: 44.395,41 u.c. Iteração: 172. Tempo de execução: 1.485,85s.



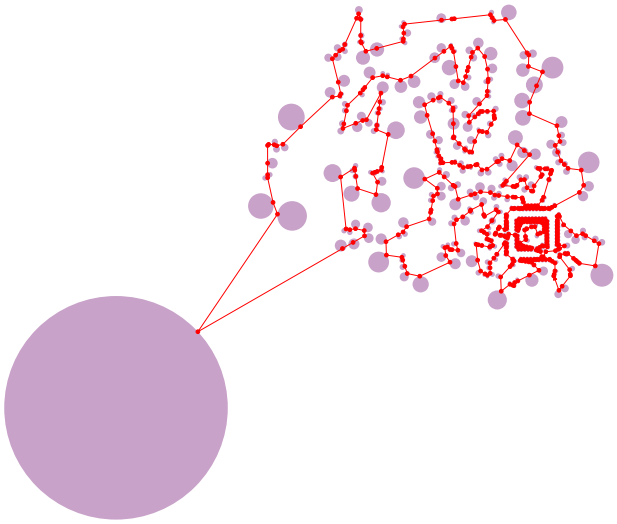
(c) Solução com InsercaoBest: 47.705,79 u.c. Iteração: 58. Tempo de execução: 3.416,85s.



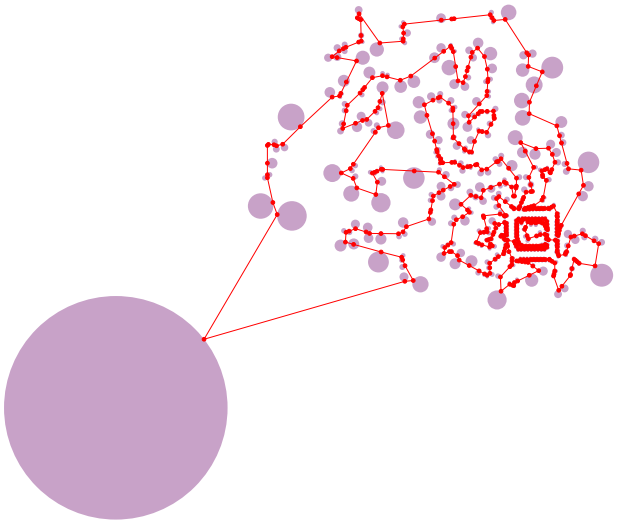
(d) Solução com InsercaoFirst: 47.409,40 u.c. Iteração: 149. Tempo de execução: 3.449,93s.

Figura 4.23: Soluções encontradas por cada um dos métodos para a instância *pr264*.

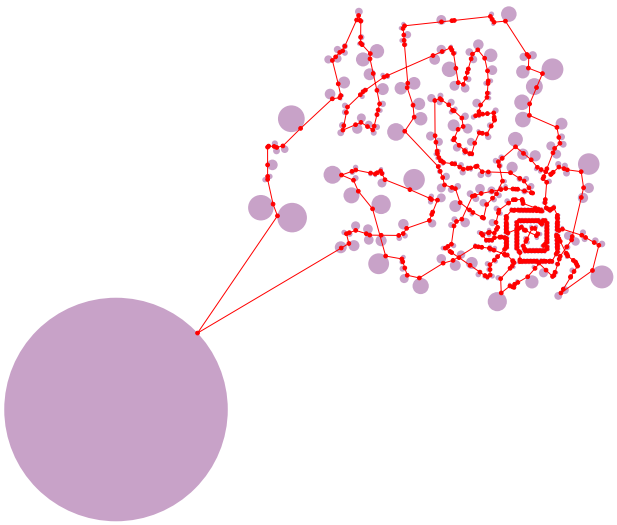
Para finalizar, exibimos as soluções encontradas para a instância *d493* na Figura 4.24.



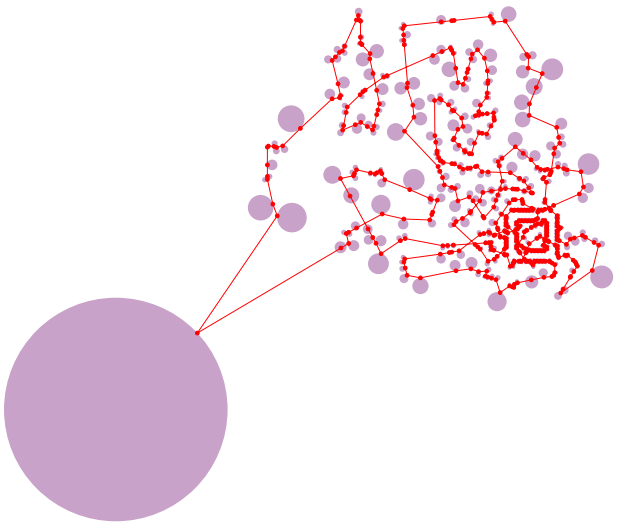
(a) Solução com 2optBest: 26.885,68 u.c. Iteração: 112. Tempo de execução: 15.518,41s.



(b) Solução com 2optFirst: 27.291,22 u.c. Iteração: 733. Tempo de execução: 18.459,19s.



(c) Solução com InsercaoBest: 28.020,69 u.c. Iteração: 216. Tempo de execução: 53.637,98s.



(d) Solução com InsercaoFirst: 28.805,17 u.c. Iteração: 947. Tempo de execução: 68.215,80s.

Figura 4.24: Soluções encontradas por cada um dos métodos para a instância *d493*.

Em resumo, a partir dos experimentos realizados com as instâncias aleatórias e da biblioteca TSPLIB, concluimos que o algoritmo 2optBest (5) encontra soluções melhores quando comparado com os outros três algoritmos propostos neste trabalho, e na maioria dos casos em menor tempo. O algoritmo 2optFirst (6) se sobressai sobre o 2optBest no quesito tempo de execução apenas para valores de m menores. Conforme aumentamos a quantidade de bolas do problema, o algoritmo 2optFirst tende a realizar um grande número de soluções até parar, consumindo mais tempo de processamento. Uma tática seria definir um número máximo de iterações mais exigente, mas isso poderia implicar na perda de qualidade da solução, tendo em vista que o decréscimo do valor funcional dos iterando tende a ser mais lento devido a estratégia do *primeiro vizinho* que melhora empregada por este algoritmo.

Observamos também que os algoritmos InsercaoBest e InsercaoFirst podem levar quatro vezes mais tempo de execução que os demais algoritmos abordados. Este comportamento já era algo esperado, já que a heurística de inserção utilizada neste algoritmo constrói uma vizinhança quase duas vezes maior que a heurística 2-Opt. Além disso, observamos que estes algoritmos necessitam de mais iterações até atingir o critério de parada. Através das figuras de soluções dos algoritmos InsercaoBest e First detectamos que uma das razões dos tours encontrados por eles terem maior comprimento está no fato de o movimento do tipo *realocação* realizado pela heurística de inserção não conseguir reordenar os pontos para desfazer todos os segmentos que se "cruzam". Assim sendo, uma estratégia seria substituir, nos Algoritmos 7 e 8, a heurística do vizinho mais próximo por outra capaz de construir tours melhores. Algumas opções seriam as heurísticas construtivas *nearest insertion*, *cheapest insertion* ou *farthest insertion*, a exemplo de [6], sendo interessante realizar novos experimentos para avaliar se com estas estratégias conseguiríamos obter soluções melhores e em tempo razoável.

Capítulo 5

Considerações finais

Nosso objetivo neste trabalho foi apresentar a formulação do problema do caixeiro viajante contínuo (CTSP) e propor algoritmos para resolver numericamente o caso em que os conjuntos visitados são bolas disjuntas do $\mathbb{R}^2$.

Nos algoritmos iterativos (5), (6), (7) e (8) propostos no Capítulo 3, combinamos a heurística construtiva do vizinho mais próximo e as heurísticas 2-Opt e de inserção com um método de busca em bloco de coordenadas (BCDM). Em cada algoritmo, a heurística NN é utilizada na construção de uma ordenação inicial T^0 e as heurísticas de busca local agem modificando essa permutação das bolas através dos seus movimentos característicos, enquanto que o método BCD é responsável pelo computo dos pontos visitados em cada bola. A atualização dos pontos pelo BCDM é realizada na ordem dada pelas permutações obtidas com as heurísticas.

Por conseguinte, experimentos numéricos foram realizados com cada um desses algoritmos para avaliar seu desempenho e comparar os resultados encontrados por cada um. Nos experimentos observamos que os algoritmos que utilizam movimentos do tipo 2-Opt se sobressaem, na maioria dos exemplos, aos métodos que usam movimentos de realocação em relação a qualidade de soluções e tempo de execução. Outro detalhe observado é que, apesar dos métodos com a estratégia do *melhor vizinho* possuírem iterações mais custosas - pois avaliam todos os vizinhos antes de definir o próximo iterando - eles encontram na maioria dos testes soluções melhores, de menor comprimento, do que os algoritmos que utilizam a estratégia do *primeiro vizinho* que melhora. Também notamos que o número de iterações tende a aumentar conforme o número de bolas cresce. Isso foi observado tanto nos experimentos com instâncias aleatórias quanto nos testes com instâncias da biblioteca TSPLIB.

Em relação ao método BCD (3) que compõe os algoritmos acima mencionados, temos que ele apresentou uma média de ciclos por chamada uniforme nos experimentos realizados com instâncias aleatórias. No entanto, este padrão não se repetiu com as instâncias da biblioteca TSPLIB, que a depender da complexidade e do número de bolas o BCDM realizou mais ciclos por chamada até

atingir o critério de parada.

Algumas propostas de trabalhos futuros para incrementar este projeto são as seguintes. Primeiro, o método BCD foi implementado com escolha cíclica de coordenadas e com o método de Newton para atualizar as coordenadas a cada iteração. No entanto, a escolha aleatória de coordenadas também poderia ser empregada neste caso e experimentos numéricos poderiam ser realizados com esta estratégia para avaliar qual é mais eficiente. Além disso, diferentes métodos heurísticos poderiam ser implementados no lugar das heurísticas NN, 2-Opt e de inserção.

Referências Bibliográficas

[1] E. Aarts and J. K. Lenstra (eds.). *Local Search in Combinatorial Optimization*. Princeton University Press, 2003. Citado 6 vezes nas páginas 6, 13, 14, 16, 17, and 18.

[2] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006. Citado 5 vezes nas páginas 6, 9, 11, 12, and 58.

[3] A. Beck and L. Tetruashvili. On de convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013. Citado na página 23.

[4] B. Behdani and J. C. Smith. An integer-programming-based approach to the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 26(3):415–432, 2014. Citado na página 21.

[5] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999. Citado 3 vezes nas páginas 21, 24, and 29.

[6] E. G. Birgin and J. M. Martínez. Block coordinate descent for smooth nonconvex constrained minimization. *Computational Optimization and Applications*, 83:1–27, 2021. Citado 4 vezes nas páginas 9, 19, 20, and 65.

[7] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Graduate School of Industrial Administration*, 1976. Citado 2 vezes nas páginas 13 and 14.

[8] W. J. Cook. In pursuit of the traveling salesman: mathematics at the limits of computation, 2012. Citado na página 11.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Algoritmos: teoria e prática*. Campus, 2012. Citado 2 vezes nas páginas 9 and 13.

[10] G. A. Croes. A method for solving traveling salesman problems. *Operations Research*, 6: 791–812, 1958. Citado na página 17.

[11] J. Dong, N. Yang, and M. Chen. Heuristic approaches for a tsp variant: The automatic meter reading shortest tour problem. In E. K. Baker, A. Joseph, A. Mehrotra, and M. A. Trick, editors, *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, pages 145–163. Springer US, Boston, MA, 2007. Citado 2 vezes nas páginas 20 and 21.

[12] M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956. Citado na página 17.

[13] M. Gendreau, A. Hertz, and G. Laporte. New insertion and post optimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1094, 1992. Citado 2 vezes nas páginas 14 and 18.

[14] M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335, 1998. Citado na página 18.

[15] F. Greco. *Travelling Salesman Problem*. I-Tech Education and Publishing KG, 2008. Citado na página 11.

[16] G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006. Citado na página 11.

[17] K. L. Hoffman, M. Padberg, and G. Rinaldi. Traveling salesman problem. In *Encyclopedia of Operations Research and Management Science*, pages 1573–1578. Springer US, 2013. Citado na página 11.

[18] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (eds). *The traveling salesman problem: A guided tour of combinatorial optimization*. Wiley: Chichester., 1985. Citado na página 11.

[19] D.G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*. International Series in Operations Research & Management Science. Springer US, 2008. ISBN 9780387745039. Citado 2 vezes nas páginas 21 and 29.

[20] F. Lukás. *The Close Enough Travelling Salesman Problem in the polygonal domain*. Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2021. Citado na página 20.

[21] J. M. Martínez and S. A. Santos. *Métodos computacionais de otimização*. Departamento de Matemática Aplicada - UNICAMP, 1995. Citado na página 29.

[22] W. K. Mennell. *Heuristics for Solving Three Routing Problems: Close-Enough Traveling Salesman Problem, Close-Enough Vehicle Routing Problem, Sequence- Dependent Team Orienteering Problem*. PhD thesis, University of Maryland (College Park, Md.), 2009. Citado na página 20.

[23] Y. E. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012. Citado 2 vezes nas páginas 22 and 23.

[24] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006. Citado 2 vezes nas páginas 21 and 29.

[25] A. D. Placido, C. Archetti, and C. Cerrone. A genetic algorithm for the close-enough traveling salesman problem with application to solar panels diagnostic reconnaissance. *Computers & Operations Research*, 145:105–831, 2022. Citado na página 21.

[26] S. Poikonen, X. Wang, and B. Golden. The vehicle routing problem with drones: Extended models and connections. *Networks*, 70(1):34–43, 2017. Citado na página 21.

[27] M. J. D. Powell. On search directions for minimization algorithms. *Mathematical Programming*, 4:193–201, 1973. Citado na página 24.

[28] G. Reinelt. TSPLIB - a traveling salesman problem library, 1991. URL `http://comopt.ifi.uni-heidelberg.de/`. Citado 4 vezes nas páginas 10, 36, 56, and 60.

[29] J. J. Robinson. On the hamiltonian game (a traveling salesman problem). *RAND Research Memorandum RM-303*, 1949. Citado na página 11.

[30] L. G. M. Santos. *Métodos de Busca em Coordenada*. Dissertação de Mestrado em Ciência da Computação, Universidade de São Paulo, Instituto de Matemática e Estatística, 2018. Citado 2 vezes nas páginas 22 and 23.

[31] R. Shuttleworth, B. L. Golden, S. Smith, and E. Wasil. Advances in meter reading: Heuristic solution of the close enough traveling salesman problem over a street network. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 487–501. Springer US, Boston, MA, 2008. Citado na página 21.

[32] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151:3–34, 2015. Citado na página 23.

Apêndice A

As tabelas a seguir apresentam os dados e soluções obtidos pelos algoritmos 2optBest (5), 2optFirst (6), InsercaoBest (7) e InsercaoFirst (8) para as instâncias aleatórias do Capítulo 4 geradas com $m = 50$, 200, 300, 400 e 500 bolas.

Instância		#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
				#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
2optBest	I1	9	41,4842099951	10.584	71.856	1.176,0	7.984,0	6,8	38,01	11,99	0,85
	I2	11	49,4468560175	12.936	98.466	1.176,0	8.951,5	7,6	35,51	14,49	1,10
	I3	12	44,1978260254	14.112	89.833	1.176,0	7.486,1	6,4	35,75	14,25	1,03
	I4	13	45,8084678819	15.288	113.892	1.176,0	8.760,9	7,4	35,60	14,40	1,25
	I5	13	43,7487664659	15.288	97.033	1.176,0	7.464,1	6,3	36,71	13,29	1,10
	I6	17	45,3128909671	19.992	131.842	1.176,0	7.755,4	6,6	35,19	14,81	1,46
	I7	13	40,4450241218	15.288	98.378	1.176,0	7.567,5	6,4	34,23	15,77	1,09
	I8	13	45,5608881209	15.288	103.055	1.176,0	7.927,3	6,7	39,22	10,78	1,17
	I9	9	46,4744980202	10.584	71.240	1.176,0	7.915,6	6,7	33,99	16,01	0,82
	I10	8	49,7050862616	9.408	75.671	1.176,0	9.458,9	8,0	35,19	14,81	0,86
Média		11,8	45,2184513877	13.876,8	95.126,6	1.176,0	8.127,1	6,9	35,94	14,06	1.07
2optFirst	I1	19	42,0749232524	7.550	53.096	397,4	2.794,5	7,0	36,10	13,90	0,65
	I2	33	47,5766774283	10.477	70.052	317,5	2.122,8	6,7	36,69	13,31	0,82
	I3	20	46,8314377833	7.727	52.133	386,4	2.606,7	6,7	34,72	15,28	0,63
	I4	41	47,7038410600	15.760	116.855	384,4	2.850,1	7,4	32,81	17,19	1,27
	I5	25	43,7898656607	8.268	50.130	330,7	2.005,2	6,1	35,71	14,29	0,61
	I6	42	46,3731709445	9.720	60.970	231,4	1.451,7	6,3	33,49	16,51	0,72
	I7	26	41,8896149889	9.010	56.422	346,5	2.170,1	6,3	34,47	15,53	0,67
	I8	43	48,7216736030	7.755	48.049	180,3	1.117,4	6,2	37,49	12,51	0,59
	I9	31	46,4894458555	9.681	64.372	312,3	2.076,5	6,6	34,79	15,21	0,76
	I10	76	47,7378618566	13.946	93.737	183,5	1.233,4	6,7	37,31	12,69	1,09
Média		35,6	45,9188512433	9.989,4	66.581,6	307,0	2.042,8	6,6	35,36	14,64	0,78
InsercaoBest	I1	16	43,4161230949	39.200	275.865	2.450,0	17.241,6	7,0	37,67	12,33	2,92
	I2	14	47,4941521478	34.300	246.302	2.450,0	17.593,0	7,2	35,56	14,44	2,58
	I3	17	46,8617298805	41.650	272.858	2.450,0	16.050,5	6,6	35,97	14,03	2,88
	I4	20	49,1204843206	49.000	364.343	2.450,0	18.217,2	7,4	33,81	16,19	3,65
	I5	13	46,2121011439	31.850	226.434	2.450,0	17.418,0	7,1	36,43	13,57	2,40
	I6	18	46,1845923317	44.100	301.210	2.450,0	16.733,9	6,8	32,67	17,33	3,08
	I7	16	45,1404007140	39.200	250.526	2.450,0	15.657,9	6,4	36,94	13,06	2,65
	I8	11	48,9729569189	26.950	179.001	2.450,0	16.272,8	6,6	37,26	12,74	1,94
	I9	17	47,6772433508	41.650	276.498	2.450,0	16.264,6	6,6	35,60	14,40	2,91
	I10	21	49,0132768207	51.450	353.631	2.450,0	16.839,6	6,9	37,38	12,62	3,67
Média		16,3	47,0093060724	39.935,0	274.666,8	2.450,0	16.828,9	6,9	35,93	14,07	2,87
InsercaoFirst	I1	32	42,2840134046	15.130	119.054	472,8	3.720,4	7,9	36,77	13,23	1,36
	I2	31	54,0161981053	30.049	214.013	969,3	6.903,6	7,1	35,24	14,76	2,28
	I3	40	48,8690005685	27.124	192.292	678,1	4.807,3	7,1	32,78	17,22	2,02
	I4	32	50,2941911433	29.528	213.313	922,8	6.666,0	7,2	32,89	17,11	2,19
	I5	18	46,2121011439	10.924	75.437	606,9	4.190,9	6,9	35,41	14,59	0,92
	I6	101	47,9606298478	37.025	263.264	366,6	2.606,6	7,1	31,36	18,64	2,70
	I7	31	45,1404007139	18.723	120.686	604,0	3.893,1	6,4	35,39	14,61	1,34
	I8	37	53,1735981033	24.327	147.451	657,5	3.985,2	6,1	37,33	12,67	1,63
	I9	36	47,6772433508	19.888	128.418	552,4	3.567,2	6,5	36,78	13,22	1,45
	I10	114	53,3244714361	39.550	262.435	346,9	2.302,1	6,6	36,33	13,67	2,79
Média		47,2	48,8951847818	25.226,8	173.636,3	617,7	4.264,2	6,9	35,03	14,97	1.87

Tabela 5.1: Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 50$ bolas.

Instância		#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
				#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
2optBest	I1	38	104,1927535330	748.638	5.019.157	19.701,0	132.083,1	6,7	134,05	65,95	174,05
	I2	44	105,3083018640	866.844	5.489.934	19.701,0	124.771,2	6,3	141,94	58,06	192,72
	I3	40	104,1352594822	788.040	4.937.659	19.701,0	123.441,5	6,3	137,94	62,06	172,79
	I4	36	101,2550406544	709.236	4.496.800	19.701,0	124.911,1	6,3	141,34	58,66	157,84
	I5	46	108,9753893238	906.246	5.756.698	19.701,0	125.145,6	6,4	142,68	57,32	202,73
	I6	41	100,9374395553	807.741	5.066.515	19.701,0	123.573,5	6,3	140,92	59,08	178,52
	I7	37	101,9409905134	728.937	4.548.556	19.701,0	122.933,9	6,2	136,05	63,95	158,64
	I8	47	106,4604469843	925.947	6.040.896	19.701,0	128.529,7	6,5	144,29	55,71	211,68
	I9	39	104,9094481089	768.339	4.777.646	19.701,0	122.503,7	6,2	133,84	66,16	166,95
	I10	40	101,4847714810	788.040	4.990.473	19.701,0	124.761,8	6,3	140,90	59,10	175,32
Média		40,8	103,9599841500	803.800,8	5.112.433,4	19.701,0	125.265,5	6,4	139,39	60,61	179,12
2optFirst	I1	171	102,4269804804	902.371	6.176.718	5.277,0	36.121,2	6,8	128,71	71,29	209,86
	I2	163	102,3812759914	918.247	5.678.678	5.633,4	34.838,5	6,2	138,11	61,89	198,27
	I3	257	110,8625573293	1.083.243	6.598.610	4.215,0	25.675,5	6,1	139,73	60,27	231,45
	I4	74	104,4216768267	620.405	3.859.505	8.383,9	52.155,5	6,2	134,63	65,37	134,54
	I5	182	107,2004033228	862.265	5.475.267	4.737,7	30.083,9	6,3	138,37	61,63	192,04
	I6	157	103,7206480839	789.503	5.183.367	5.028,7	33.015,1	6,6	133,65	66,35	179,26
	I7	167	105,5145994504	990.811	6.354.631	5.933,0	38.051,7	6,4	140,66	59,34	221,14
	I8	271	105,8467142735	873.040	5.830.569	3.221,5	21.515,0	6,7	137,98	62,02	203,47
	I9	296	106,0458464839	1.003.490	6.132.701	3.390,2	20.718,6	6,1	136,52	63,48	213,61
	I10	166	103,9776879753	1.477.896	8.885.227	8.903,0	53.525,5	6,0	142,34	57,66	311,16
Média		190,4	105,2398390218	952.127,1	6.017.527,3	5.472,3	34.570,0	6,3	137,07	62,93	209,48
InsercaoBest	I1	76	108,8515782768	3.024.800	21.671.851	39.800,0	285.155,9	7,2	134,77	65,23	730,38
	I2	68	119,5722209948	2.706.400	17.456.732	39.800,0	256.716,6	6,5	138,75	61,25	598,13
	I3	65	119,0964103066	2.587.000	17.322.318	39.800,0	266.497,2	6,7	138,97	61,03	590,08
	I4	54	105,1449805525	2.149.200	13.988.844	39.800,0	259.052,7	6,5	141,49	58,51	479,88
	I5	70	114,6701537460	2.786.000	18.013.235	39.800,0	257.331,9	6,5	145,54	54,46	621,07
	I6	49	110,9589123274	1.950.200	13.086.941	39.800,0	267.080,4	6,7	135,79	64,21	446,77
	I7	60	114,7003851053	2.388.000	15.692.436	39.800,0	261.540,6	6,6	139,10	60,90	537,69
	I8	67	122,4912176829	2.666.600	18.314.093	39.800,0	273.344,7	6,9	133,36	66,64	615,81
	I9	83	114,6691164398	3.303.400	22.246.968	39.800,0	268.035,8	6,7	137,50	62,50	754,58
	I10	47	115,0323855511	1.870.600	12.806.109	39.800,0	272.470,4	6,8	131,13	68,87	433,51
Média		63,9	114,5187360983	2.543.220,0	17.059.952,7	39.800,0	266.722,6	6,7	137,64	62,36	580,79
InsercaoFirst	I1	156	114,0137826792	1.364.221	9.410.305	8.745,0	60.322,5	6,9	132,12	67,88	318,94
	I2	218	115,9678773395	2.311.315	14.716.479	10.602,4	67.506,8	6,4	137,66	62,34	503,51
	I3	191	123,5136743367	1.987.327	12.859.277	10.404,9	67.326,1	6,5	138,25	61,75	440,54
	I4	153	102,4044886911	2.709.835	18.239.767	17.711,3	119.214,2	6,7	135,19	64,81	615,19
	I5	174	116,1432024484	2.113.518	14.552.565	12.146,7	83.635,4	6,9	138,67	61,33	500,84
	I6	233	113,1266979253	1.363.952	9.583.809	5.853,9	41.132,2	7,0	133,25	66,75	327,37
	I7	240	114,0894133429	1.853.137	12.395.767	7.721,4	51.649,0	6,7	131,94	68,06	419,98
	I8	201	123,3410353843	1.340.644	9.041.115	6.669,9	44.980,7	6,7	132,41	67,59	308,10
	I9	234	121,0835401125	2.039.742	13.619.994	8.716,8	58.205,1	6,7	133,17	66,83	463,34
	I10	198	114,6798201210	1.933.115	13.389.920	9.763,2	67.625,9	6,9	132,38	67,62	453,79
Média		199,8	115,8363532381	1.901.680,6	12.780.899,8	9.833,5	66.159,8	6,7	134,50	65,50	435,16

Tabela 5.2: Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 200$ bolas.

Instância		#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
				#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
2optBest	I1	65	126,6164002278	2.895.815	17.908.069	44.551,0	275.508,8	6,2	206,22	93,78	915,89
	I2	66	129,8895757683	2.940.366	19.127.058	44.551,0	289.803,9	6,5	204,05	95,95	975,58
	I3	59	129,1475639370	2.628.509	15.864.874	44.551,0	268.896,2	6,0	220,25	79,75	823,05
	I4	56	126,9400281691	2.494.856	15.994.705	44.551,0	285.619,7	6,4	205,38	94,62	814,45
	I5	49	123,3278241680	2.182.999	14.069.768	44.551,0	287.138,1	6,4	199,95	100,05	715,76
	I6	62	129,3384305259	2.762.162	16.629.527	44.551,0	268.218,2	6,0	203,65	96,35	844,97
	I7	76	125,3899672859	3.385.876	20.067.757	44.551,0	264.049,4	5,9	213,53	86,47	1.032,12
	I8	65	125,2608726368	2.895.815	17.980.965	44.551,0	276.630,2	6,2	209,51	90,49	917,36
	I9	42	126,7352182115	1.871.142	11.465.962	44.551,0	272.999,1	6,1	204,94	95,06	589,77
	I10	50	123,3420235819	2.227.550	14.307.635	44.551,0	286.152,7	6,4	200,47	99,53	727,84
Média		59,0	126,5987904512	2.628.509,0	16.341.632,0	44.551,0	277.501,6	6,2	206,79	93,21	835.68
2optFirst	I1	240	129,6768208445	2.360.934	14.997.512	9.837,2	62.489,6	6,4	197,12	102,88	757,91
	I2	387	127,2359525123	4.045.389	24.725.557	10.453,2	63.890,3	6,1	209,92	90,08	1.269,40
	I3	250	130,0787197288	2.334.337	14.046.232	9.337,3	56.184,9	6,0	212,58	87,42	726,14
	I4	180	133,4198077232	3.171.901	20.858.922	17.621,7	115.882,9	6,6	194,91	105,09	1.046,35
	I5	197	130,4866581382	3.479.725	22.449.232	17.663,6	113.955,5	6,5	199,01	100,99	1.126,55
	I6	257	130,3022448811	3.847.091	23.075.737	14.969,2	89.788,9	6,0	204,52	95,48	1.176,24
	I7	262	127,9800150337	2.290.099	13.672.603	8.740,8	52.185,5	6,0	200,48	99,52	698,41
	I8	180	131,5574499909	2.263.328	14.336.408	12.574,0	79.646,7	6,3	198,12	101,88	724,32
	I9	224	129,2381051278	1.857.882	11.569.494	8.294,1	51.649,5	6,2	201,27	98,73	593,85
	I10	260	126,9237302268	5.013.068	32.092.551	19.281,0	123.432,9	6,4	197,97	102,03	1.611,87
Média		243,7	129,6899504207	3.066.375,4	19.182.424,8	12.877,2	80.910,7	6,2	201,59	98,41	973,10
InsercaoBest	I1	73	144,5429799258	6.548.100	42.950.566	89.700,0	588.363,9	6,6	203,74	96,26	2.158,07
	I2	117	143,6967228250	10.494.900	74.702.099	89.700,0	638.479,4	7,1	202,98	97,02	3.728,16
	I3	80	137,1263464822	7.176.000	47.596.871	89.700,0	594.960,9	6,6	206,56	93,44	2.393,76
	I4	100	140,9654353948	8.970.000	59.118.980	89.700,0	591.189,8	6,6	207,58	92,42	2.977,49
	I5	55	133,4378039341	4.933.500	33.345.229	89.700,0	606.276,9	6,8	203,20	96,80	1.679,22
	I6	74	140,1246491962	6.637.800	42.649.801	89.700,0	576.348,6	6,4	207,66	92,34	2.149,24
	I7	86	136,1655397183	7.714.200	51.677.552	89.700,0	600.901,8	6,7	201,44	98,56	2.585,11
	I8	82	145,9799114339	7.355.400	49.216.172	89.700,0	600.197,2	6,7	204,03	95,97	2.462,59
	I9	85	136,6206837975	7.624.500	48.923.335	89.700,0	575.568,7	6,4	204,96	95,04	2.461,59
	I10	92	134,2320033564	8.252.400	56.089.882	89.700,0	609.672,6	6,8	200,03	99,97	2.791,22
Média		84,4	139,2892076064	7.570.680,0	50.627.048,7	89.700,0	598.196,0	6,7	204,22	95,78	2.538,64
InsercaoFirst	I1	239	142,8583397451	6.036.083	39.555.545	25.255,6	165.504,4	6,6	210,03	89,97	2.005,38
	I2	484	143,3654893179	15.231.208	105.807.528	31.469,4	218.610,6	6,9	207,77	92,23	5.280,78
	I3	285	135,8440812910	6.114.963	40.134.107	21.456,0	140.821,4	6,6	206,22	93,78	2.017,72
	I4	286	141,4638102511	5.688.853	40.082.160	19.891,1	140.147,4	7,0	196,16	103,84	1.991,36
	I5	228	134,1695758235	7.421.693	51.294.260	32.551,3	224.974,8	6,9	201,35	98,65	2.555,00
	I6	389	144,9190496188	10.202.252	64.794.897	26.226,9	166.567,9	6,4	205,15	94,85	3.244,42
	I7	407	137,1810189322	6.854.812	45.271.584	16.842,3	111.232,4	6,6	201,69	98,31	2.278,20
	I8	369	148,2599111750	7.018.509	45.707.267	19.020,3	123.867,9	6,5	203,44	96,56	2.291,88
	I9	180	137,2222226931	6.049.974	40.316.292	33.611,0	223.979,4	6,7	199,85	100,15	2.019,24
	I10	328	133,2232192591	6.089.119	41.618.811	18.564,4	126.886,6	6,8	194,08	105,92	2.070,13
Média		319,5	139,8506718107	7.670.746,6	51.458.245,1	24.488,8	164.259,3	6,7	202,57	97,43	2.575,41

Tabela 5.3: Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 300$ bolas.

Instância		#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
				#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
2optBest	I1	83	167,7308691773	6.590.283	40.605.082	79.401,0	489.217,8	6,2	273,69	126,31	2.731,63
	I2	74	168,7895901365	5.875.674	36.175.195	79.401,0	488.854,0	6,2	278,77	121,23	2.448,63
	I3	79	166,9842137587	6.272.679	39.345.777	79.401,0	498.047,8	6,3	280,46	119,54	2.666,13
	I4	67	164,1919725400	5.319.867	33.165.813	79.401,0	495.012,1	6,2	268,65	131,35	2.231,43
	I5	71	174,3206930351	5.637.471	35.981.380	79.401,0	506.780,0	6,4	273,28	126,72	2.420,68
	I6	74	171,4789490675	5.875.674	37.593.958	79.401,0	508.026,5	6,4	267,20	132,80	2.512,53
	I7	66	175,4968884083	5.240.466	32.734.562	79.401,0	495.978,2	6,2	276,50	123,50	2.211,39
	I8	87	167,7817549964	6.907.887	41.639.538	79.401,0	478.615,3	6,0	288,24	111,76	2.837,11
	I9	84	170,0840011907	6.669.684	42.522.813	79.401,0	506.223,9	6,4	273,41	126,59	2.857,03
	I10	85	165,1525089378	6.749.085	43.317.818	79.401,0	509.621,4	6,4	265,77	134,23	2.898,05
Média		77,0	169,2011441248	6.113.877,0	38.308.193,6	79.401,0	497.637,7	6,3	274,60	125,40	2.581,46
2optFirst	I1	387	168,3936667977	6.763.631	42.638.550	17.477,1	110.177,1	6,3	274,65	125,35	2.870,07
	I2	330	169,5571519051	5.663.920	35.958.563	17.163,4	108.965,3	6,3	278,17	121,83	2.428,12
	I3	429	171,2543065261	7.947.374	52.347.090	18.525,3	122.021,2	6,6	268,76	131,24	3.503,00
	I4	203	168,4170880205	5.640.861	34.664.261	27.787,5	170.759,9	6,1	270,74	129,26	2.346,44
	I5	313	176,7123178680	7.550.098	47.086.268	24.121,7	150.435,4	6,2	276,49	123,51	3.187,35
	I6	417	174,8201255153	6.948.449	42.741.963	16.662,9	102.498,7	6,2	271,54	128,46	2.869,75
	I7	378	181,4078361798	5.470.057	34.186.928	14.471,0	9.0441,6	6,2	276,28	123,72	2.316,02
	I8	385	176,1611445424	9.687.583	58.825.855	25.162,6	152.794,4	6,1	269,61	130,39	3.938,27
	I9	475	171,4282859695	8.087.581	51.871.387	17.026,5	109.202,9	6,4	267,85	132,15	3.448,19
	I10	447	169,2822348370	9.390.918	57.072.332	21.008,8	127.678,6	6,1	266,57	133,43	3.825,01
Média		376,4	172,7434158161	7.315.047,2	45.739.319,7	19.940,7	124.497,5	6,3	272,07	127,93	3.073,22
InsercaoBest	I1	115	183,0767089518	18.354.000	121.921.819	159.600,0	1.060.189,8	6,6	275,83	124,17	8.143,64
	I2	106	187,3238429061	16.917.600	111.361.709	159.600,0	1.050.582,2	6,6	283,58	116,42	7.479,44
	I3	140	188,3750736792	22.344.000	154.227.382	159.600,0	1.101.624,1	6,9	275,52	124,48	10.265,60
	I4	118	179,4950082715	18.832.800	125.718.352	159.600,0	1.065.409,8	6,7	274,89	125,11	8.403,38
	I5	116	188,2208335587	18.513.600	127.285.342	159.600,0	1.097.287,5	6,9	268,79	131,21	8.395,11
	I6	114	188,6402501328	18.194.400	124.186.112	159.600,0	1.089.351,9	6,8	268,98	131,02	8.196,44
	I7	108	187,1414742368	17.236.800	113.881.684	159.600,0	1.054.460,0	6,6	276,64	123,36	7.599,01
	I8	160	186,9704776112	25.536.000	163.561.405	159.600,0	1.022.258,8	6,4	284,44	115,56	10.925,99
	I9	109	193,4975499574	17.396.400	121.093.848	159.600,0	1.110.952,8	7,0	270,25	129,75	8.028,25
	I10	113	178,5362369587	18.034.800	123.007.118	159.600,0	1.088.558,6	6,8	265,13	134,87	8.133,70
Média		119,9	186,1277456264	19.136.040,0	128.624.477,1	159.600,0	1.074.067,5	6,7	274,40	125,60	8.557,06
InsercaoFirst	I1	486	186,1201457513	24.454.014	161.385.277	50.316,9	332.068,5	6,6	273,69	126,31	10.729,19
	I2	491	185,6294876366	10.463.226	69.877.535	21.310,0	142.316,8	6,7	279,63	120,37	4.674,71
	I3	571	186,0980116546	17.742.885	123.163.786	31.073,4	215.698,4	6,9	269,02	130,98	8.161,81
	I4	430	177,9647981378	14.042.004	96.945.636	32.655,8	225.455,0	6,9	265,68	134,32	6.432,19
	I5	419	195,4280575485	17.389.374	121.699.720	41.502,1	290.452,8	7,0	264,89	135,11	8.068,17
	I6	436	193,2221161851	24.255.337	161.982.107	55.631,5	371.518,6	6,7	264,22	135,78	10.649,42
	I7	473	188,2999889909	18.220.115	121.358.767	38.520,3	256.572,5	6,7	269,30	130,70	8.056,13
	I8	490	190,4472852512	13.482.346	88.465.283	27.515,0	180.541,4	6,6	268,91	131,09	5.873,08
	I9	473	188,5594277176	16.445.734	113.627.810	34.769,0	240.227,9	6,9	268,18	131,82	7.495,43
	I10	503	192,5629581483	16.805.510	117.294.432	33.410,6	233.189,7	7,0	262,74	137,26	7.713,14
Média		477,2	188,4332277022	17.330.054,5	117.580.035,3	36.670,5	248.804,1	6,8	268,63	131,37	7.785,33

Tabela 5.4: Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 400$ bolas.

Instância		#iter	Comprimento final do tour	Uso total do BCDM		Uso por iter do BCDM		BCDM por chamada			Tempo
				#chamadas	#ciclos	#chamadas/iter	#ciclos/iter	#ciclos/chamada	Newton/ciclo	Projeção/ciclo	
2optBest	I1	90	183,1124849328	11.182.590	67.014.807	124.251,0	744.609,0	6,0	354,98	145,02	5.767,30
	I2	96	185,3839629292	11.928.096	70.130.502	124.251,0	730.526,1	5,9	358,44	141,56	6.057,78
	I3	85	190,3976087204	10.561.335	68.124.746	124.251,0	801.467,6	6,5	328,72	171,28	5.665,97
	I4	97	190,9588728446	12.052.347	76.450.277	124.251,0	788.147,2	6,3	339,33	160,67	6.358,04
	I5	87	189,3617439946	10.809.837	67.900.777	124.251,0	780.468,7	6,3	337,26	162,74	5.649,63
	I6	99	189,7648243626	12.300.849	74.830.852	124.251,0	755.867,2	6,1	352,05	147,95	6.261,54
	I7	103	186,0504151594	12.797.853	79.540.919	124.251,0	772.241,9	6,2	348,88	151,12	6.626,46
	I8	109	182,2959780130	13.543.359	82.985.445	124.251,0	761.334,4	6,1	354,91	145,09	6.945,70
	I9	92	183,2536011978	11.431.092	69.918.055	124.251,0	759.978,9	6,1	346,78	153,22	5.862,31
	I10	98	188,6750404481	12.176.598	74.378.472	124.251,0	758.964,0	6,1	344,09	155,91	6.214,14
Média		95,6	186,9254532602	11.878.395,6	73.127.485,2	124.251,0	765.360,5	6,2	346,54	153,46	6.140,89
2optFirst	I1	396	188,6525460031	16.988.764	101.020.751	42.900,9	255.102,9	5,9	348,23	151,77	8.406,68
	I2	511	189,7224204701	16.899.945	101.042.085	33.072,3	197.734,0	6,0	340,90	159,10	8.452,13
	I3	578	189,6186372347	12.972.017	83.118.567	22.442,9	143.803,7	6,4	334,07	165,93	6.897,63
	I4	554	193,5636682164	16.627.065	101.583.218	30.012,8	183.363,2	6,1	335,11	164,89	8.459,09
	I5	471	193,9650519840	10.575.136	67.514.895	22.452,5	143.343,7	6,4	339,59	160,41	5.622,25
	I6	613	192,6797644138	16.551.933	101.475.068	27.001,5	165.538,4	6,1	338,37	161,63	8.442,53
	I7	391	190,5153032079	10.772.245	68.557.135	27.550,5	175.337,9	6,4	335,97	164,03	5.673,69
	I8	441	188,4275555350	12.965.651	81.198.144	29.400,6	184.122,8	6,3	334,52	165,48	6.740,02
	I9	457	183,4540750774	11.289.083	68.138.979	24.702,6	149.100,6	6,0	340,83	159,17	5.699,38
	I10	407	194,7507618276	12.566.573	75.460.414	30.876,1	185.406,4	6,0	336,58	163,42	6.273,10
Média		481,9	190,5349783970	13.820.841,2	84.910.925,6	29.041,3	178.285,4	6,2	338,42	161,58	7.066,65
InsercaoBest	I1	148	212,1731095546	36.926.000	238.165.530	249.500,0	1.609.226,6	6,4	356,02	143,98	19.874,47
	I2	158	205,8184866471	39.421.000	256.108.155	249.500,0	1.620.937,7	6,5	347,01	152,99	21.284,74
	I3	149	205,3246992828	37.175.500	250.682.180	249.500,0	1.682.430,7	6,7	332,77	167,23	20.607,88
	I4	146	208,5410150785	36.427.000	247.148.051	249.500,0	1.692.794,9	6,8	335,69	164,31	20.304,96
	I5	129	207,2511148322	32.185.500	210.537.287	249.500,0	1.632.072,0	6,5	342,17	157,83	17.459,99
	I6	146	210,3799299408	36.427.000	244.124.283	249.500,0	1.672.084,1	6,7	342,86	157,14	20.183,61
	I7	165	206,8059234425	41.167.500	277.895.581	249.500,0	1.684.215,6	6,8	338,06	161,94	22.800,71
	I8	129	199,4034743605	32.185.500	213.871.597	249.500,0	1.657.919,4	6,6	346,15	153,85	17.776,97
	I9	159	202,9357015715	39.670.500	262.593.744	249.500,0	1.651.533,0	6,6	348,59	151,41	21.823,84
	I10	131	200,2807188123	32.684.500	220.021.346	249.500,0	1.679.552,3	6,7	335,07	164,93	18.123,49
Média		146,0	205,8914173523	36.427.000,0	242.114.775,4	249.500,0	1.658.276,6	6,6	342,44	157,56	20.024,07
InsercaoFirst	I1	453	214,2744865947	29.009.676	189.533.912	64.039,0	418.397,2	6,5	344,07	155,93	15.663,20
	I2	510	209,7928179656	45.869.907	310.887.377	89.941,0	609.583,1	6,8	335,94	164,06	25.699,66
	I3	809	205,3273145034	34.944.739	235.660.550	43.195,0	291.298,6	6,7	335,82	164,18	19.373,64
	I4	550	212,3095460368	36.720.051	254.579.212	66.763,7	462.871,3	6,9	319,86	180,14	20.641,25
	I5	501	205,1337125046	28.246.231	183.939.055	56.379,7	367.143,8	6,5	343,07	156,93	15.239,25
	I6	605	212,5463692092	32.296.331	218.997.943	53.382,4	361.980,1	6,8	339,55	160,45	18.091,60
	I7	785	207,1996494823	33.697.839	228.490.019	42.927,2	291.070,1	6,8	344,20	155,80	18.838,84
	I8	513	203,9051666867	31.232.798	204.857.491	60.882,6	399.332,3	6,6	343,91	156,09	16.973,71
	I9	456	202,7681260400	31.605.704	206.311.468	69.310,8	452.437,4	6,5	349,08	150,92	17.228,55
	I10	608	205,5937965842	25.596.481	171.114.705	42.099,5	281.438,7	6,7	328,39	171,61	14.030,63
Média		579,0	207,8850985607	32.921.975,7	220.437.173,2	5.8892,1	393.555,3	6,7	338,39	161,61	18.178,03

Tabela 5.5: Desempenho dos Algoritmos 5 a 8 para cada uma das 10 instâncias com $m = 500$ bolas.