Henrique Alves Magalhães

O Problema do Ladrão Viajante
Propriedades e Heurísticas

Brasil

2016, 30 de novembro

Henrique Alves Magalhães

O Problema do Ladrão Viajante
Propriedades e Heurísticas

Dissertação sobre o estudo do Problema do Ladrão Viajante de forma qualitativa e quantitativa, apresentada na Universidade Federal de Minas Gerais

Universidade Federal de Minas Gerais – UFMG

Instituto de Ciências Exatas

Programa de Pós-Graduação em Matemática

Orientador: Ricardo Hiroshi Caldeira Takahashi

Brasil

2016, 30 de novembro

Henrique Alves Magalhães

O Problema do Ladrão Viajante
Propriedades e Heurísticas

Dissertação sobre o estudo do Problema do Ladrão Viajante de forma qualitativa e quantitativa, apresentada na Universidade Federal de Minas Gerais

Brasil, 30 de novembro de 2016:

Ricardo Hiroshi Caldeira Takahashi
Orientador

Professor
Convidado 1

Professor
Convidado 2

Brasil
2016, 30 de novembro

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Os agradecimentos principais são direcionados à minha mãe Nilce Alves, meu padrasto André Luis, e meus irmãos Sabrina e Rodrigo, além de todos aqueles que contribuíram para minha formação acadêmica, como meus professores João Batista e Maria Aparecida Lage que acreditaram em mim e em meu potencial antes mesmo que eu acreditasse.

Agradecimentos especiais são direcionados aos meus padrinhos Iraci Corte e Terezinha Silveira Corte, que sempre me apoiaram e me aconselharam, e também a meus amigos Ricardo Souza, Samuel Quirino, Cristiano Benjamin e Luis Paiva pelas discussões sobre os mais diversos tópicos, inclusive sobre o tema deste trabalho, dos quais abriram minha mente e a completaram com muitas idéias.

Por fim, agradeço a meu orientador Ricardo Takahashi e por sua orientação e conselhos, me permitindo fazer este estudo tão gratificante.

"Não é o conhecimento, mas o ato de aprender,
não a posse mas o ato de chegar lá,
que concede a maior satisfação."
(Carl Friedrich Gauss)

Resumo

Muitos problemas clássicos são estudados em otimização devido à suas capacidades de modelar algumas classes de problemas do mundo real, além de sua própria complexidade. Mas problemas clássicos de referência como o Problema da Mochila e o Problema do Caixeiro Viajante são os mesmos há algumas décadas, com algumas variantes sendo desenvolvidas ao longo deste período.

Em 2013 foi proposto um problema que é baseado nestes dois, e foi chamado de Problema do Ladrão Viajante. Este problema não é apenas uma variação destes dois problemas clássicos, mas interconecta diversas características destes dois problemas sendo mais complexo e assim, possibilita uma melhor modelagem de problemas reais modernos.

Foi feito um algoritmo genético com operadores personalizados, e este foi comparado a um método de resolução proposto que se baseia na divisão do problema em clusters. Esta divisão busca simplificar o problema e reduzir o espaço de busca com o fim de se obter um melhor desempenho do algoritmo genético original. Para esta comparação são feitas simulações de problemas onde os clusters se tornam cada vez mais próximos espacialmente, de modo que estes agrupamentos se tornam cada vez menos evidentes.

Palavras-chaves: Problema do Caixeiro Viajante, Problema da Mochila, Problema do Ladrão Viajante,Otimização Combinatória, Clusterização.

Abstract

Many classic problems are studied in optimization because of their ability to model some classes of real-world problems, in addition to its own complexity. But classic reference problems like the Knapsack Problem and the Travelling Salesman Problem are the same for decades, with some variants being developed over this period.

In 2013 it was proposed a problem which is based on these two, and was called the Travelling Thief Problem. This problem is not just a variation of these two classic problems, but interconnects various characteristics of these two problems being more complex and thus allow better modelling of modern real problems.

A genetic algorithm with custom operators was made, and it was compared to resolution method which is based on the clustering of the problem. This division seeks to simplify the problem and reduce the search space in order to get a better performance of the original genetic algorithm. For this comparison it was made simulations of problems where the clusters are becoming closer spatially, so that these groups are becoming less evident.

Key-words: Travelling Salesman Problem, Knapsack Problem, Travelling Thief Problem, Combinatorial optimization, Clustering.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introducao e Revisão Bibliográfica

"A Otimização é o campo de conhecimentos cujas técnicas visam determinar os extremos (máximos ou mínimos) de funções, em domínios determinados.", [1]. Esta definição mais geral permite ao leitor entender o foco desta dissertação de uma forma mais simples. A Otimização é uma área de pesquisa importante para a sociedade moderna, estudando a possibilidade de redução de custos de produção de diversos produtos, a maximização da eficiência de dispositivos ou mesmo a redução da probabilidade de ocorrência de falhas. Assim, é natural imaginar a otimização como uma ferramenta que busque melhorar, de alguma forma, processos existentes em situações reais (apesar desta não se limitar a isto), e para tanto são necessárias técnicas de modelagem matemática destes problemas de modo que as técnicas de otimização possam ser aplicadas.

Devido à grande variedade de problemas reais, são estudadas várias abordagens para estes problemas. De modo geral, as variáveis em um problema de otimização podem ser contínuas (podendo assumir valores dentro de intervalos) ou discretas (podendo assumir valores em conjuntos enumeráveis), e muitas vezes o que determina o tipo de variáveis que um dado problema de otimização possui são as características inerentes ao problema estudado, e a forma com a qual o problema foi modelado. Alguns exemplos de problemas estudados na literatura são a programação de sensores em redes sem fio utilizando programação linear [2], e o estudo de problemas de restauração de energia na rede elétrica [3].

Dentre os problemas estudados na literatura, se destacam os *problemas clássicos*. Estes problemas são vastamente estudados devido ao fato de muitos problemas reais serem modelados (às vezes de forma mais simplificada) como tais problemas. Dois problemas clássicos muito importantes e amplamente estudados na literatura são o Problema do Caixeiro Viajante [4] [5], que já possui diversas variantes [6], e o Problema da Mochila [7], [8] que além de ser amplamente aplicado, tem vários algoritmos e heurísticas diferentes para sua resolução [9] [10] [11].

Existem várias heurísticas e algoritmos que buscam a resolução de problemas combinatoriais como estes, como as bio-inspiradas colônias de formigas [12] e algoritmos

genéticos [9] ou métodos não bio-inspirados como a incorporação de problemas combinatórios estruturados em árvores em espaços de Hilbert [13], assim como diversas outras meta-heurísticas [14].

Para [15] há "uma distância crescente entre pesquisa e prática" quanto à aplicabilidade de meta-heurísticas para problemas reais. Como a complexidade de problemas reais cresce constantemente, e problemas de referência (como os dois citados neste texto) são os mesmos nos últimos 50 anos, cada vez mais estes não são suficientes para descrever os problemas atuais. Tendo isto em mente, foi proposto o Problema do Ladrão Viajante [15] com o intuito de se obter um novo problema de referência a ser estudado, e que seja complexo o suficiente para melhor modelar problemas reais. Este problema foi baseado no Problema do Caixeiro Viajante e no Problema da Mochila, interconectando características de ambos, de modo a criar um problema novo.

Este problema foi estudado também por [16] que propõe um algoritmo de busca local para este problema, e o compara com um algoritmo de busca local aleatória e com um algoritmo evolutivo. Porém por ser um problema novo, este problema não foi estudado produndamente por muitos outros pesquisadores.

A abordagem via busca local de problemas combinatórios já existe na literatura [17], e baseando-se nesta meta-heurística, baseando-se também no agrupamento de elementos do problema em clusters ([18]) foi criada e estudada uma heurística que visa simplificar o Problema do Ladrão Viajante em um problema mais simples, com o objetivo de se obter um melhor desempenho nos algoritmos de resolução do mesmo. Para isto, foi feito um algoritmo genético com operadores personalizados ao problema, e foram comparados os resultados obtidos pelo algoritmo genético aplicado ao problema original com os resultados obtidos pelo algoritmo aplicado ao problema simplificado. Foram feitas várias simulações de alguns problemas de modo a se comparar os resultados obtidos e determinar quando esta heurística se mostrará melhor que um algoritmo que não se utiliza de nenhuma informação prévia sobre o problema, ou sequer considera suas características e propriedades.

Organização do texto

Este texto é composto por sete capítulos enumerados, além de suas referências.

No Capítulo 1 é feita uma retrospectiva sobre alguns tópicos de otimização, como trabalhos feitos e problemas estudados. Neste capítulo é feita a revisão bibliográfica deste estudo, além de introduzir de forma resumida o que foi feito neste estudo.

No Capítulo 2 é apresentada a fundamentação teórica necessária para o entendimento deste estudo. São enunciados os problemas clássicos TSP e KP, além de descrever o TTP1, problema estudado neste texto. Também é feita uma introdução a algoritmos genéticos e seus operadores.

No Capítulo 3 é apresentada a fundamentação metodológica necessária para o entendimento da metodologia utilizada. São estudados os efeitos da manipulação de soluções na função-objetivo, introduzindo os operadores genéticos utilizados no algoritmo genético construído.

No Capítulo 4 é construído o algoritmo genético utilizado. É feita uma descrição completa de seus operadores, além de se mostrar um exemplo da aplicação do algoritmo.

No Capítulo 5 é introduzida a principal contribuição deste estudo: a simplificação do TTP1 via *clusters*. Neste capítulo é dada a definição e a aplicação dos clusters neste problema, e é construída uma adaptação do algoritmo genético do capítulo anterior utilizando esta abordagem.

No Capítulo 6 são apresentados os resultados obtidos nas simulações de diversos problemas utilizando os dois algoritmos construídos nos capítulos (4) e (5). Neste capítulo é feita a comparação e discussão dos resultados obtidos por estes algoritmos.

No Capítulo 7 é feita a conclusão deste estudo. Neste capítulo é feito um resumo do que foi apresentado neste texto, apresentando as conclusões obtidas pelos resultados do Capítulo 6, além de se traçar os próximos passos a serem dados em estudos futuros.

Além disso, cada capítulo (com exceção do Capítulo 1) é iniciado com uma breve descrição do que será apresentado em cada uma de suas seções. Por fim, as referências bibliográficas utilizadas neste estudo são apresentadas ao final do texto, logo após o Capítulo 7.

2 Fundamentação Teórica

Este capítulo trata dos elementos técnicos que compõem o Problema do Ladrão Viajante, bem como as principais ferramentas e conceitos necesários para o entendimento do que foi feito. No texto que se segue, denota-se o Problema do Ladrão Viajante apenas por TTP (sigla em inglês para, *Traveling Thief Problem*).

Na seção 2.1 serão expostos os enunciados dos problemas clássicos que inspiraram o TTP bem como seus parâmetros.

Na seção 2.2 enuncia-se o TTP, expondo sua complexidade bem como suas diferenças em relação aos dois problemas clássicos nos quais este foi inspirado. Um modelo específico para o TTP é então enunciado (e denotado TTP1) e exemplificado, de modo a demonstrar sua complexidade e suas diferenças em relação aos problemas de origem.

Na seção 2.3 é feita a análise do TTP1 como problema combinatório, avaliando qual o número possível de soluções a serem avaliadas a fim de se encontrar a melhor das soluções. Um exemplo é apresentado utilizando os dados do exemplo da seção 2.2.

Na seção 2.4 é apresentada a estrutura básica de algoritmos genéticos, introduzindo possíveis operadores de mutação e cruzamento de soluções do TTP1 que servirão como base para os operadores utilizados no algoritmo genético construído.

As notações utilizadas em todos os demais capítulos deste trabalho seguem o mesmo padrão de [15], que enuncia duas versões para o problema.

2.1 Problemas Clássicos de Otimização

Em computação, muitos problemas inspiraram o desenvolvimento de diversos algoritmos e se tornaram bastante conhecidos como *problemas clássicos*, e dentre os diversos problemas existentes, o Problema do Caixeiro Viajante (*Traveling Salesman Problem*, ou TSP) e o Problema da Mochila (*Knapsack Problem*, ou KP) são dois problemas NP-difíceis bastante famosos e serviram de inspiração para o TTP. Seguem abaixo os seus enunciados.

O Problema do Caixeiro Viajante

Considere um conjunto $C = \{1, 2, ..., n\}$ de n cidades, e seja $D = \{d_{\{i,j\}}\}$ a matriz de distâncias entre estas cidades, de modo que $d_{i,j}$ é a distância entre as cidades i e j. Um caixeiro precisa visitar estas cidades, de modo que todas as cidades de C sejam visitadas exatamente uma vez num menor tempo possível. Supõe-se que a velocidade v_c do caixeiro é constante, e portanto este busca encontrar o caminho de menor comprimento. A função objetivo será:

$$f(\bar{x}) = \sum_{i=1}^{n-1} t_{i,i+1} + t_{n,1}, \bar{x} = (x_1, ..., x_n) \tag{2.1}$$

onde $\bar{x}$ representa o caminho do caixeiro, com $x_i \in C, i = 1, \ldots, n$ e $x_i \neq x_j$ se $i \neq j$. $t_{a,b}$ representa o tempo necessário para percorrer a distância d_{x_a,x_b}, ou seja

$$t_{a,b} = \frac{d_{x_a,x_b}}{v_c} \tag{2.2}$$

Logo, a Equação 2.1 calcula o tempo necessário para que o caixeiro percorra as cidades $\{x_1, x_2, ..., x_{n-1}, x_n, x_1\}$ nesta ordem. O objetivo final é minimizar $f(\bar{x})$.

O Problema da Mochila

Seja $I = \{I_1, I_2, ..., I_m\}$ uma coleção de m itens, sendo que cada item I_j possui duas propriedades: p_j e w_j. p_j representa o valor (em unidades monetárias) do j-ésimo objeto, enquanto w_j representa seu peso (também nas devidas unidades de peso). Dada uma mochila com capacidade de carga W, deseja-se selecionar um conjunto de itens para se colocar nesta mochila, de modo que o valor total dos itens selecionados seja máximo, mas sua carga máxima não seja excedida, ou melhor:

$$\text{maximizar } g(\bar{y}) = \sum_{i=1}^{m} p_i y_i, \bar{y} = (y_1, ..., y_m) \tag{2.3}$$

$$\text{sujeito a } \sum_{i=1}^{m} w_i y_i \leq W \tag{2.4}$$

onde $y_i \in \{0, 1\}$ com $y_i = 1$ indicando que o i-ésimo objeto foi escolhido, e $y_i = 0$ caso contrário. $g(\bar{y})$ é o valor total dos itens escolhidos, de modo que o objetivo final é maximizar g de modo que a Equação 2.4 seja respeitada.

2.2 O Problema do Ladrão Viajante

Tomando como base os problemas precedentes, o TTP é definido: Seja $C = \{1, 2, ..., n\}$ um conjunto de n elementos, denotados como *cidades*, e seja $D = \{d_{ij}\}$ a matriz de distâncias correspondente. Seja $I = \{I_1, I_2, ..., I_m\}$ uma coleção de m itens que estão distribuídos dentre estas cidades, sendo que cada item possui *valor* e *peso* (essas propriedades denotadas como no KP). Considere agora um ladrão que deseja visitar cada cidade exatamente uma vez, recolhendo alguns destes itens, e colocando-os numa mochila. A mochila deste ladrão possui uma carga máxima W que não pode ser excedida. Este problema pode ser organizado da seguinte forma:

Parâmetros de entrada:

- Disponibilidade do item I_i em cada cidade representado por $A_i \subseteq C \setminus \{\varnothing\}$, de modo que cada item deve estar disponível em pelo menos uma cidade.
- Demais parâmetros do TTP, que dependerão de como o problema foi definido (número de cidades e suas conexões, velocidade do ladrão, pesos e valores dos objetos e etc.) bem como da interação destes parâmetros que definirão o objetivo final da otimização. Esta última interação pode ser definida de diversas formas, de modo que os dois formatos definidos em [15] serão expostos a seguir.

Parâmetros de saída:

- A solução do TTP, que é da forma $(\bar{x}, \bar{z})$, onde $\bar{x} = (x_1, ..., x_n)$ é um caminho, como definido no TSP, e $\bar{z} = (z_1, ..., z_m)$ com $z_i \in \{0 \cup A_i\}$ $\forall i$ representando o *plano de roubo*, de modo que $z_i = k$ indica que o i-ésimo objeto será retirado da cidade k, enquanto que $z_i = 0$ indica que o i-ésimo objeto não deve ser retirado.

Enquanto no KP, busca-se maximizar o valor dos itens coletados, respeitando a carga máxima da mochila, enquanto que no TSP busca-se minimizar o tempo gasto pelo

caixeiro - ou ladrão no caso do TTP - para que todas as cidades sejam visitadas. Assim, no TTP exitem dois subproblemas, que são o de achar o melhor plano de retirada de objetos $\bar{z}$ e o de achar o menor caminho $\bar{x}$, e uma forma de conectar estes dois subproblemas geraria um novo problema a ser estudado. Dentre as mais diversas formas, [15] desenvolveu as duas formas que são denotadas como TTP1 e TTP2. O primeiro modelo foi o objeto de estudo deste trabalho, e seu enunciado é mostrado a seguir com um exemplo simples.

O Modelo TTP1

Neste modelo, busca-se maximizar o lucro final do ladrão, de modo que os dois subproblemas passarão a ter uma interdependência impondo-se as seguintes condições:

- A velocidade do ladrão não é mais considerada constante durante todo tipo de percurso, mas depende diretamente do peso da mochila que ele carrega. Assim, dada uma velocidade máxima v_{max} e uma velocidade mínima v_{min}, que correspondem respectivamente à velocidade do ladrão quando a mochila está vazia ($z_i = (0, 0, ..., 0) \Rightarrow \sum_{i=1}^{m} w_i z_i = 0$) e à velocidade do ladrão quando a mochila está completamente cheia ($\sum_{i=1}^{m} w_i z_i = W$). Neste modelo, a velocidade do ladrão é definida pela equação $v_c = v_{max} - W_c \frac{v_{max} - v_{min}}{W}$, onde v_c é a velocidade corrente e W_c é o peso corrente da mochila.

- A mochila do ladrão é *alugada*, de modo que após percorrer o trajeto escolhido, ele devolveria a mochila para o locador pagando uma taxa de locação. Esta taxa é de um valor $\$R$ por unidade de tempo, de modo que o lucro final do ladrão é dado por:

$$G(\bar{x}, \bar{z}) = g(\bar{z}) - R \times f(\bar{x}, \bar{z}) \tag{2.5}$$

  onde G é o lucro final do ladrão, $g(\bar{z}) = \sum_{i=1}^{m} p_i z_i$ é a soma dos valores dos objetos recolhidos, e f é o tempo total gasto no percurso.

Assim, o objetivo do TTP1 é **maximizar** G. Assim, o tempo do percurso foi vinculado ao peso dos objetos através da variação da velocidade durante o percurso. Da mesma forma, o valor dos objetos retirados foi vinculado ao tempo do percurso através da taxa de aluguel.

Este novo problema aparenta ser mais complexo que o KP e o TSP, mas ele surge da necessidade de estudos de problemas mais completos (e consequentemente mais complexos)

e que modelam melhor problemas reais. Intuitivamente, pode ser feita a analogia de um caminhão que deve percorrer diversos pontos de coleta de produtos e retorná-los à uma base responsável pelo seu armazenamento final. Este caminhão gastaria mais energia para se locomover se estiver com muita carga, ou mesmo se locomoveria mais lentamente dependendo do caso. Neste sistema real, ao invés de se modelar o problema pelo TSP, ou mesmo pelo KP, é mais razoável usar o TTP1 por ser mais completo e permitir a inserção de um maior número de parâmetros de interesse. Um exemplo teórico simples enunciado em [15] é mostrado a seguir, de modo a expor a interdependência entre os dois subproblemas citados.

TTP1: Exemplo Simples

Suponha um TTP1 com os seguintes parâmetros:

- $n = 4$ (quatro cidades), $m = 5$ (cinco tipos de objetos), $W = 3$ (capacidade máxima da mochila), $v_{máx} = 1$ e $v_{min} = 0.1$ (velocidades máxima e mínima do ladrão, respectivamente).
- Matriz de distâncias $D = \begin{pmatrix} 0 & 5 & 6 & 6 \\ 5 & 0 & 5 & 6 \\ 6 & 5 & 0 & 4 \\ 6 & 6 & 4 & 0 \end{pmatrix}$
- Parâmetros (p, w) (valor e peso, respectivamente) dos itens I_i: $I_1 = (100, 3), I_2 = (40, 1), I_3 = (40, 1), I_4 = (20, 2), I_5 = (30, 3)$.
- Disponibilidade A_i de cada item nas cidades, indicando em quais cidades o i-ésimo item está disponível, sendo: $A_1 = \{3\}, A_2 = \{3\}, A_3 = \{3\}, A_4 = \{2, 4\}$, e $A_5 = \{2\}$. A matriz de disponibilidade destes itens é definida como $A = \{a_{i,j}\}$ com $a_{i,j} = 1$ se o item i está disponível na cidade j, e $a_{i,j} = 0$ caso contrário, ou seja:

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{2.6}$$

- Taxa de aluguel $R = \$1$ por unidade de tempo.

Este exemplo é ilustrado pela Figura 1.

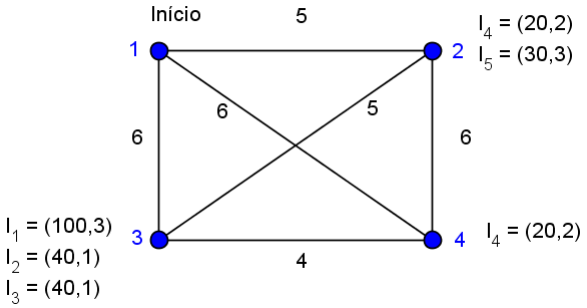


Figura 1 – Exemplo de um TTP1 simples.

Uma possível solução para este TTP1 é $\bar{x} = (1, 3, 2, 4, 1)$, indicando a ordem em que as cidades serão visitadas pelo ladrão, e $\bar{z} = (0, 3, 0, 2, 0)$, de modo que os objetos 1,, 3 e 5 não foram retirados de quaisquer cidades, enquanto o objeto 2 foi retirado da cidade 3, e o objeto 4 foi retirado da cidade 2. A validade desta solução pode ser verificada observando que $3 \in A_2$, e $2 \in A_4$.

O valor da função objetivo desta solução pode ser calculado da seguinte forma:

- Inicialmente é percorrida a distância $d_{1,3}$ sem carga, pois não foi retirado nenhum objeto da cidade 1, e portanto a velocidade neste percurso será $v_c = v_{máx} = 1$, e o tempo gasto será $t_{1,3} = \frac{d_{1,3}}{v_{máx}} = \frac{6}{1} = 6$.
- No percurso seguinte, é percorrida a distância $d_{3,2} = 5$ mas agora com carga $W_c = 1$ pois o objeto 2 foi retirado na cidade 3, e portanto a velocidade será $v_c = 1 - 1 \times \frac{1-0}{3} = 0.7$. Logo o tempo gasto nesta parte do percurso é $t_{3,2} = \frac{d_{3,2}}{v_c} = \frac{5}{0.7} = 7.14$.
- A seguir, é percorrida a distância $d_{2,4} = 6$ com o peso $W_c = 1 + 2 = 3$, pois o objeto 4 é retirado da cidade 2. Assim, a velocidade do ladrão será $v_{mín}$, tendo em vista que $W_c = W$, implicando num tempo $t_{2,4} = \frac{d_{2,4}}{v_{mín}} = \frac{6}{0.1} = 60$.
- Finalmente, a distância $d_{4,1} = 6$ é percorrida também na velocidade mínima, pois apesar de nenhum objeto ter sido retirado na cidade 4 (o que é impedido pelo fato de a mochila se encontrar cheia), a carga carregada continua sendo a carga máxima, implicando num tempo $t_{4,1} = \frac{d_{4,1}}{v_{mín}} = \frac{6}{0.1} = 60$.

Assim, o tempo total necessário será $f(\bar{x}, \bar{z}) = 6 + 7.14 + 60 + 60 = 133.14$ unidades de tempo. O valor total dos itens retirados será $g(\bar{z}) = p_2 + p_4 = 40 + 20 = 60$, e a função objetivo (lucro do ladrão) será de $G(\bar{x}, \bar{z}) = g(\bar{z}) - R \times f(\bar{x}, \bar{y}) = 60 - 1 \times 133.14 = -73.14$, indicando que o ladrão teria um prejuízo se escolhesse esta solução.

A solução ótima foi previamente calculada por [15], bem como as soluções dos subproblemas isolados sendo:

- Solução para o problema do menor *tour* (caminho mínimo): $\bar{x} = (1, 2, 3, 4, 1)$ e $\bar{x} = (1, 4, 3, 2, 1)$.
- Solução para o problema do melhor plano de roubo (lucro máximo): $\bar{z} = (3, 0, 0, 0, 0)$.
- Melhor solução para o TTP1 enunciado: $(\bar{x}, \bar{z}) = \{(1, 2, 4, 3, 1), (0, 3, 3, 0, 0)\}$ com $G(\bar{x}, \bar{z}) = 50$.

De fato, não é de se surpreender que a solução do problema seja diferente das soluções de seus subproblemas combinadas: $(\bar{x}, \bar{z}) = \{(1, 2, 3, 4, 1), (3, 0, 0, 0, 0)\}$ e $(\bar{x}, \bar{z}) = \{(1, 4, 3, 2, 1)), (3, 0, 0, 0, 0)\}$. A interdependência entre os dois subproblemas que foi construída é o fator causador deste fenômeno, o que motiva o estudo do TTP1 como um novo problema, e não como um caso específico do KP ou do TSP.

2.3 Análise do TTP1 como um problema combinatório

Nesta seção será feito o estudo do TTP1 de forma combinatória, avaliando quantas soluções são possíveis para avaliação pela função objetivo, a fim de se encontrar a melhor solução.

Tome inicialmente o exemplo dado na seção anterior, consistindo de um TTP1 com n = 4, m = 5, e matriz A de disponibilidade de cada item como dada na Equação 2.6.

Neste exemplo, cada solução é da forma:
$\{(1, \sigma(2), \sigma(3), \sigma(4), 1), (\alpha(1), \alpha(2), ..., \alpha(5))\}$, onde $\sigma(i) \neq \sigma(j)\ \forall i \neq j$, com $i, j \in \{2, ..., n\}$, e $\alpha(k) \in \{0\} \bigcup \{i \mid a_{k,i} \neq 0\}$ de modo que $\alpha(k)$ indica se o k-ésimo objeto foi retirado de alguma cidade válida (elementos unitários $a_{k,i}$ de A) ou se este objeto não foi retirado (escolhando $\alpha(k) = 0$).

Observando a matriz A, tem-se que o objeto $\{4\}$ se encontra em duas cidades diferentes, enquanto os demais objetos estão disponíveis em apenas uma das cidades. Assim, existem $2 \times 2 \times 2 \times 3 \times 3 \times 2 = 48$ escolhas diferentes para $\bar{z}$. Algumas destas escolhas não são factíveis, pois por exemplo: $\bar{z} = (3, 3, 3, 2, 2)$ (que é a retirada de todos os objetos) não satisfaz a condição de peso máximo, pois $W_c = 3 + 1 + 1 + 2 + 3 = 10 > W$. De fato, há 12 escolhas de $\bar{z}$ factíveis que são mostradas na seção 2.3 abaixo:

$$\begin{aligned}
\bar{z_1} &= (0, \quad 0, \quad 0, \quad 0, \quad 0) \\
\bar{z_2} &= (3, \quad 0, \quad 0, \quad 0, \quad 0) \\
\bar{z_3} &= (0, \quad 3, \quad 0, \quad 0, \quad 0) \\
\bar{z_4} &= (0, \quad 3, \quad 3, \quad 0, \quad 0) \\
\bar{z_5} &= (0, \quad 3, \quad 0, \quad 2, \quad 0) \\
\bar{z_6} &= (0, \quad 0, \quad 3, \quad 0, \quad 0) \\
\bar{z_7} &= (0, \quad 0, \quad 3, \quad 2, \quad 0) \\
\bar{z_8} &= (0, \quad 0, \quad 0, \quad 2, \quad 0) \\
\bar{z_9} &= (0, \quad 0, \quad 0, \quad 4, \quad 0) \\
\bar{z_{10}} &= (0, \quad 3, \quad 0, \quad 4, \quad 0) \\
\bar{z_{11}} &= (0, \quad 0, \quad 3, \quad 4, \quad 0) \\
\bar{z_{12}} &= (0, \quad 0, \quad 0, \quad 0, \quad 2)
\end{aligned}$$

Tabela 1 – Possíveis valores de $\bar{z}$ no exemplo da seção 2.2.

Já os possíveis valores de $\bar{x}$ serão todas as permutações de $\{2, ..., n\}$ precedidas por um elemento $\{1\}$, e seguidas por outro elemento $\{1\}$, a saber:

$$\begin{aligned}
\bar{x_1} &= (1, \quad 2, \quad 3, \quad 4, \quad 1) \\
\bar{x_2} &= (1, \quad 2, \quad 4, \quad 3, \quad 1) \\
\bar{x_3} &= (1, \quad 3, \quad 2, \quad 4, \quad 1) \\
\bar{x_4} &= (1, \quad 3, \quad 4, \quad 2, \quad 1) \\
\bar{x_5} &= (1, \quad 4, \quad 2, \quad 3, \quad 1) \\
\bar{x_6} &= (1, \quad 4, \quad 3, \quad 2, \quad 1)
\end{aligned}$$

Tabela 2 – Possíveis valores de $\bar{x}$ no exemplo da seção 2.2.

Assim, há 6 possíveis valores para $\bar{x}$, de modo que uma possível solução do problema exemplificado será da forma $(\bar{x_i}, \bar{z_j})$, onde $i \in \{1, ..., 6\}$ e $j \in \{1, ..., 12\}$, resultando em $6 \times 12 = 96$ soluções possíveis. Portanto, o número de soluções possíveis cresce em razão de n, m e A, tornando o número de possibilidades muito alto para problemas complexos.

Suponha agora um TTP1 consistindo de n cidades e m objetos. Suponha também A é dada tal que não há valores infactíveis de $\bar{z}$. A existência de tal problema tal que a matriz A não impeça certas possibilidades para $\bar{z}$ é garantida se for tomado um TTP1 tal que $\sum_{i=1}^{m} w_i \leq W$. Seja $q \in \mathbb{N}^*$ tal que $|A_i| \geq q \ \forall i$, ou seja, cada objeto se encontra em

pelo menos q cidades, então há pelo menos $(n-1)! \times (q+1)^m$ possíveis soluções para o TTP1.

A complexidade do TSP da ordem de $\frac{(n-1)!}{2}$[1] foi incrementada ao adicionar o KP e uma interdependência entre estes dois problemas, de modo que o TTP1 tem uma complexidade ainda maior de resolução.

2.4 O Algoritmo Genético

O TTP1 é um problema recente que ainda não foi abordado por heurísticas específicas que levem em conta sua estrutura, de modo que se justifica o uso de um algoritmo de "uso geral" como um *algoritmo genético* para estudá-lo. Algoritmos genéticos são definidos em [2] como "métodos de pesquisa probabilísticos inspirados nos princípios da seleção natural e da genética", de modo que estes processam elementos dentro do espaço de busca (um conjunto de possíveis soluções) a fim de *evoluir* este conjunto de soluções até a obtenção de soluções com qualidade elevada. Neste modelo bioinspirado, cada solução é representada por um conjunto de **cromossomos**, sendo que cada cromossomo é composto por **genes**. Esta população de soluções é transformada de modo a:

- Selecionar as soluções mais aptas para que estas sejam as progenitoras da próxima geração. Este processo de **seleção** tipicamente é probabilístico.

- Operar as soluções selecionadas de modo a gerar uma próxima geração (população de soluções).

Assim, sempre que uma nova geração é gerada a partir dos progenitores selecionados de uma população inicial, esta nova geração deve obedecer ao principio (bioinspirado) **hereditariedade**, de modo que a nova geração deva herdar características existentes em seus progenitores. Logo, dada uma população inicial $P_0 = \{p_{1,0}, ..., p_{m,0}\}$ de m soluções, o operador de seleção $s(P_i) = \{p_{\alpha_1,i}, ..., p_{\alpha_n,i}\}$, com $\alpha_k \in \{1, ..., m\}\ \forall k \in \{1, ..., m\}$ e $n \leq m$, seleciona n progenitores a partir de m candidatos.

[1] O fator 2 aparece dividindo $(n-1)!$ porque no TSP busca-se um ciclo composto pelos n vértices, de modo que soluções da forma $(x_1, x_2, ..., x_{n-1}, x_n)$ são equivalentes à soluções representadas como $(x_n, x_{n-1}, ..., x_2, x_1)$.

Um operador de cruzamento $c_{k,l}(\{p_{1,i}, ..., p_{k,i}\}) = \{p_{1,i+1}, ..., p_{l,i+1}\}$ gera l soluções a partir de um conjunto de k progenitores, de modo que cada solução $p_{j,i+1}$ da nova geração (i+1) herdou características (conjuntos de genes) de soluções da geração i anterior. Neste cruzamento, a possibilidade de **mutação genética** é considerada, de modo que cada solução da nova geração possui uma chance de sofrer mutação $m(p_{j,i+1} = p'_{j,i+1})$ (modificação de seu código genético). O objetivo do operador de mutação é aumentar o espaço de busca, a fim de evitar que as próximas gerações fiquem presas em mínimos locais associados às gerações anteriores.

Um modelo de algoritmo genético é exemplificado no algoritmo 1 abaixo.

Algoritmo 1: Modelo de um algoritmo genético

```
1 t = 0
2 Gerar população inicial P_t
3 critério de parada não for satisfeito Selecionar os indivíduos mais aptos como
  progenitores
4 Cruzar os progenitores (com mutação), obtendo uma nova população P_{t+1}
5 t = t+1
6 Avaliar os indivíduos com a função objetivo F(P_t)
```

A escolha dos operadores de cruzamento e mutação, a chance de mutação, e o critério de parada do algoritmo irão definir sua eficácia, de modo que estas escolhas afetam a qualidade das soluções encontradas, bem como a velocidade de convergência (se o algoritmo convergir). A seguir serão mostradas as escolhas de operadores estudados neste trabalho, exemplificando suas aplicações e justificando suas escolhas.

2.4.1 Operadores de Cruzamento

Algoritmos genéticos são caracterizados pela representação de suas soluções como cromossomos formados por *genes*. Uma boa representação das soluções que permita ao algoritmo trabalhar com seus principais parâmetros é importante, e a representação utilizada neste estudo é mostrada na subseção 2.4.1 onde uma solução $(\bar{x_1}, \bar{z_2})$ é formada pelos dois cromossomos $\bar{x_1}$ e z_2, sendo os dois cromossomos representados no formato já conhecido das soluções, mas lembrando que seus genes, apesar de representados por números naturais, são elementos **diferentes** por pertencerem a diferentes conjuntos, sendo que os genes do *Cromossomo 1* são cidades representadas por elementos do conjunto $\{1, ..., n\}$ enquanto os genes do *Cromossomo 2* são objetos representados por elementos

do conjunto $\{1, ..., m\} \bigcup \{0\}$ (foi utilizada uma representação representação alternativa de modo a destacar cada gene).

$\bar{x_1} =$ | 1 | 2 | 3 | 4 | 1 | $\quad \bar{z_2} =$ | 0 | 0 | 0 | 0 | 0 |

Tabela 3 – Representação de uma solução e seus dois cromossomos $\bar{x_1}$ (Cromossomo 1) e $\bar{z_2}$ (Cromossomo 2).

Existem diversos operadores de cruzamento na literatura e aplicáveis às mais diversas representações de soluções. Porém, o TTP1 possui diversas peculiaridades inerentes ao problema e à representação típicas de suas soluções. Tendo em mãos a seção 2.3 e a Tabela 2.3 observe que as soluções $(\bar{x_1}, \bar{z_2})$ e $(\bar{x_2}, \bar{z_2})$ diferem apenas nas posições dos genes 3 e 4 que estão trocados. Defina $t_{(\bar{x_i}, \bar{z_j})}$ como o tempo gasto no trajeto da solução $(\bar{x_i}, \bar{z_j})$ e v_θ como sendo a velocidade do ladrão, dado que sua mochila possui os itens $\theta \subset \{1, ..., m\} \bigcup \{0\}$. Calcula-se f para estas duas soluções: $t_{(\bar{x_1}, \bar{z_2})} = f(\bar{x_1}, \bar{z_2}) = 5 + 5 + \frac{4+6}{v_{\{1\}}} = 5 + 5 + \frac{4+6}{0.1} = 110$ unidades de tempo, e $t_{(\bar{x_2}, \bar{z_2})} = f(\bar{x_1}, \bar{z_2}) = 5 + 6 + 4 + \frac{5}{v_{\{1\}}} = 5 + 6 + 4 + \frac{5}{0.1} = 65$ unidades de tempo. Note que esta pequena diferença entre as duas soluções resultou numa grande diferença nos tempos de percurso, e como ambas possuem $\bar{z_2}$ como segundo cromossomo, ambas possuem o mesmo valor de g, sendo $g(\bar{z_2} = 100)$. O valor da função-objetivo para as duas soluções será:

$$G(\bar{x_1}, \bar{z_2}) = g(z_2) - R \times f(\bar{x_1}, \bar{z_2}) = 100 - 1 \times 110 = -10 \tag{2.7}$$

$$G(\bar{x_2}, \bar{z_2}) = g(z_2) - R \times f(\bar{x_1}, \bar{z_2}) = 100 - 1 \times 65 = 35 \tag{2.8}$$

Assim, a mera troca de dois genes no Cromossomo 1 mantendo todos os demais genes de ambos os cromossomos constante pode causar uma grande diferença no valor das funções-objetivo (repare que a melhor solução mostrada em Figura 2.2 tem $G(\bar{x}, \bar{z}) = 50$).

Outros operadores como a *recombinação com um ponto de corte* mostrada em [2] não são diretamente aplicáveis devido à natureza do Cromossomo 1 que lhe dá uma estrutura composta por permutações. Foi então identificada a necessidade da escolha de um operador personalizado para este problema.

Uma *métrica* que indique a *distância* entre duas soluções dadas, ou uma forma de dizer "*quais soluções se encontram*" entre "*duas soluções quaisquer*" seriam úteis para a

definição deste novo operador. Assim, suponha duas soluções $(\bar{x_1}, \bar{z_1})$ e $(\bar{x_5}, \bar{z_1})$, com $\bar{x_1}$ e $\bar{x_5}$ definidas na subseção 2.4.1. Considere uma terceira solução $(\bar{x_2}, \bar{z_1})$, e repare que a partir de trocas feitas entre termos adjacentes no primeiro cromossomo é possível partir de uma solução para outra, como mostrado na Equação 2.4.1.

$\bar{x_1} =$	1	2	**3**	**4**	1	$\bar{z_1} =$	0	0	0	0	0
$\bar{x_2} =$	1	**2**	**4**	3	1	$\bar{z_1} =$	0	0	0	0	0
$\bar{x_5} =$	1	4	2	3	1	$\bar{z_1} =$	0	0	0	0	0

Tabela 4 – Obtendo $(\bar{x_5}, \bar{z_1})$ a partir de $(\bar{x_1}, \bar{z_1})$ com trocas de genes adjacentes do Cromossomo 1.

Assim, é possível obter $(\bar{x_2}, \bar{z_1})$ a partir de $(\bar{x_1}, \bar{z_1})$ trocando as posições dos genes 4 e 3, e da mesma forma, obtém-se $(\bar{x_5}, \bar{z_1})$ a partir de $(\bar{x_2}, \bar{z_1})$ trocando-se a posição dos genes 2 e 4. Repare que todas as três soluções mostradas prossuem o mesmo valor de $\bar{z}$, de modo que todas as trocas ocorreram no Cromossomo 1 de cada solução. Para fins de cruzamento de soluções, pode-se tomar $(\bar{x_2}, \bar{z_1})$ como solução filha das soluções $(\bar{x_1}, \bar{z_1})$ e $(\bar{x_5}, \bar{z_1})$, de modo que esta herda características das soluções progenitoras como as arestas percorridas no grafo[2] bem como quais objetos foram recolhidos e onde foram recolhidos[3].

Apesar de serem herdadas características das soluções progenitoras, ainda existem diferenças entre a solução filha (como o valor da função-objetivo aplicada a esta solução), mas vale notar que este método não garante a existência de uma solução intermediária à duas soluções progenitoras dadas, bastando notar que não há soluções intermediárias a $(\bar{x_1}, \bar{z_1})$ e $(\bar{x_2}, \bar{z_1})$, pois diferem apenas por uma troca de genes adjacentes. Da mesma forma, é possível que existam mais de uma solução filha de um casal progenitor.

Quanto à mutação, como esta exige a variação dos genes mutados, uma troca de genes em $\bar{x}$ pode ser usada como mutação deste cromossomo, de modo que $\bar{x_2}$ seria uma mutação de $\bar{x_1}$ se os genes escolhidos para a operação forem os genes 3 e 4. Da mesma forma $\bar{x_3}$ e $\bar{x_6}$ são possíveis mutações para $\bar{x_1}$. Vale ressaltar que toda operação feita em $\bar{x}$ (incluindo a mutação) pode ser escrita como uma composição deste operador básico de troca de genes adjacentes - excluindo o primeiro e último genes que devem ser fixados - devido ao fato de serem permutações de $\{2, ..., n\}$.

O $\bar{z}$ de cada solução pode ser mutado substituindo um gene não-nulo por zero, pois sabe-se que cada um de seus genes pode assumir este valor, mas melhor ainda seria

[2] $d_{\{1,2\}}$ e $d_{\{3,4\}}$ que também foram percorridas em $\bar{x_1}$, enquanto $d_{\{2,4\}}$ que foi percorrida em $\bar{x_5}$.
[3] Pois o $\bar{z}$ de cada uma das três soluções é o mesmo.

substituir um gene por um outro valor válido, que deve ser consultado na matriz A. Assim, se o i-ésimo gene for escolhido para mutação, escolhe-se um valor em $A_1 \bigcup \{0\}$ que seja diferente do valor anterior do gene. Segundo este operador, a solução $(\bar{x_5}, \bar{z_2})$ pode ser uma mutação de $(\bar{x_5}, \bar{z_1})$, de modo que o primeiro gene foi mutado.

Existem várias formas de se escolher operadores de mutação e cruzamento para este problema, sendo que estas devam explorar as propriedades inerentes ao TTP1. Os operadores construidos para implementação de um algoritmo genético que busca encontrar a solução ótima do TTP1 serão apresentados no capítulo a seguir, de modo que suas propriedades serão avaliadas e exemplos serão apresentados.

3 Fundamentação Metodológica

Neste capítulo serão estudadas as ferramentas utilizadas para a construção do algorítmo genético utilizado.

Na seção 3.1 é feita a análise do efeito (sobre a função-objetivo) da troca de duas cidades $\bar{x}(i)$ e $\bar{x}(j)$ de uma solução $\bar{x}$ do TSP, ou melhor, do TTP1 com R = \$1 e supondo uma matriz $m \times n$ A associada ao problema seja a matriz nula $A = (a_{i,j})$, com $a_{i,j} = 0$ $\forall i \in \{1, ..., m\}$, e $\forall j \in \{1, ..., n\}$.

Na seção 3.2 é feita a análise do efeito da mesma troca, mas agora considerando um TTP1 genérico, de modo a avaliar o efeito da troca de cidades de $\bar{x}$ considerando um $\bar{z}$ arbitrário tal que $(\bar{x}, \bar{z})$ é solução do TTP1.

Na seção 3.3 são enunciados os operadores utilizados no algoritmo genético estudado, tendo as seções seção 3.1 e seção 3.2 como referência para sua concepção e suas características. Nesta seção também é dado um exemplo de aplicação do operador de cruzamento obtido, e são enunciados possíveis operadores de mutação para o algoritmo genético utilizado no capítulo seguinte.

3.1 Efeito da troca de duas cidades em soluções do TSP.

Suponha um TTP1 composto por n cidades, matriz de distâncias $D = (d_{i,j})$, m objetos, e uma taxa de \$R por unidade de tempo. Suponha agora que A é nula, ou seja, nenhum objeto está disponível em quaisquer cidades. Neste caso, toda solução será da forma $(\bar{x}, \bar{z}) = (\bar{x}, (0, 0, ..., 0)^1)$, ou seja, a função-objetivo será da forma $G(\bar{x}, \bar{z}) = g(\bar{z}) - R \times f(\bar{x}, \bar{z})) = -R \times (\sum_{k=1}^{n-1} t_{k,k+1} + t_{n,1}) = -R \times \frac{\sum_{l=1}^{n-1} d_{l,l+1} + d_{n,1}}{v_{máx}}$. Ou seja, a função objetivo depende apenas da distância total a ser percorrida que depende apenas de $\bar{x}$.

[1] Vetor de m entradas nulas.

$\bar{x} =$	x_1	x_2	...	x_{i-1}	x_i	x_{i+1}	...	x_{j-1}	x_j	x_{j+1}	...	x_{n-1}	x_n	x_1
$\bar{x'} =$	x_1	x_2	...	x_{i-1}	x_j	x_{i+1}	...	x_{j-1}	x_i	x_{j+1}	...	x_{n-1}	x_n	x_1

Tabela 5 – Duas soluções semelhantes para o TTP1 sem objetos.

Sem perda de generalidade, seja $\bar{x}$ uma solução qualquer deste problema tal que:

$$\bar{x} = (x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ..., x_{j-1}, x_j, x_{j+1}, ..., x_{n-1}, x_n, x_1), i < j \text{ e } (x_1 = 1) \quad (3.1)$$

Tome agora uma solução $\bar{x'}$, com:

- $x'\bar{(i)} = \bar{x}(j)$
- $x'\bar{(j)} = \bar{x}(i)$
- $x'\bar{(k)} = \bar{x}(k)$ se $k \neq i$ ou $k \neq j$

Estas duas soluções são mostradas na Equação 3.1. Repare que as duas soluções possuem poucas arestas distintas. De fato, (supondo que $i < j - 2$) enquanto em $\bar{x}$ as arestas {i-1, i}, {i, i+1}, {j-1, j}, e {j, j+1}, são percorridas, enquanto em $\bar{x'}$ estas arestas não são percorridas, mas no lugar destas são percorridas {i-1, j}, {j, i+1}, {j-1, i}, e {i, j+1}, e todas as demais arestas são percorridas em ambas as soluções.

A Figura 2 mostra um exemplo de um caminho composto por 10 cidades, de modo que as cidades 2 e 10 terão a sua ordem de visita trocada, obtendo a Figura 3. Nesta nova figura, as arestas {3,10}, {10,9}, {7,2} e {2,5} foram percorridas, enquanto as arestas {3,10}, {10,9}, {7,2} e {2,5} deixaram de ser percorridas.

Figura 2 – Caminho $\bar{x}$ antes da troca da ordem das cidades 2 e 10.

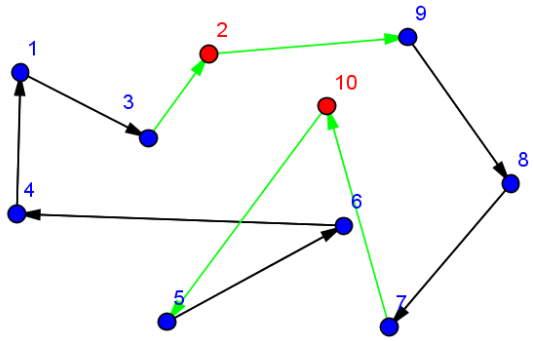
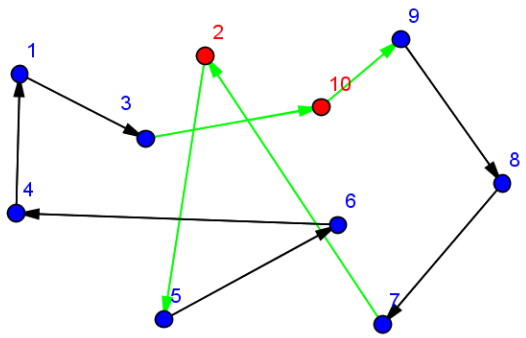


Figura 3 – Caminho $\bar{x'}$ obtido pela troca das cidades 2 e 10.



Assim, 4 novas arestas serão percorridas enquanto 4 arestas deixam de ser percorridas. Caso $j = i + 2$ ou $j = i + 1$, serão 2 arestas excluídas e duas arestas incluídas ao caminho original, como exemplificado nas figuras (4), (5), (6), e (7).

Figura 4 – Caminho $\bar{x}$ antes da troca da ordem das cidades 2 e 8.

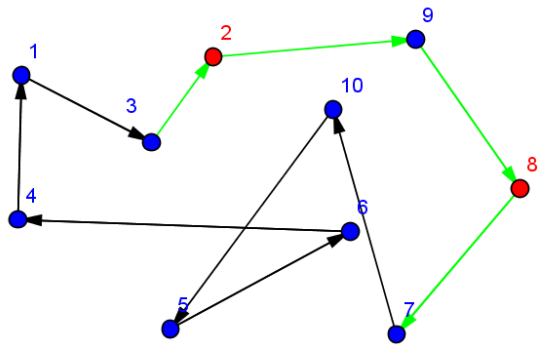


Figura 5 – Caminho $\bar{x'}$ obtido pela troca das cidades 2 e 8.

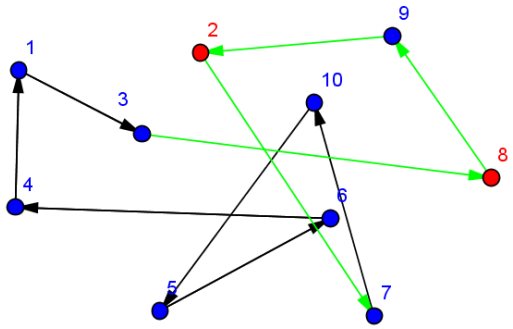


Figura 6 – Caminho $\bar{x}$ antes da troca da ordem das cidades 2 e 9.

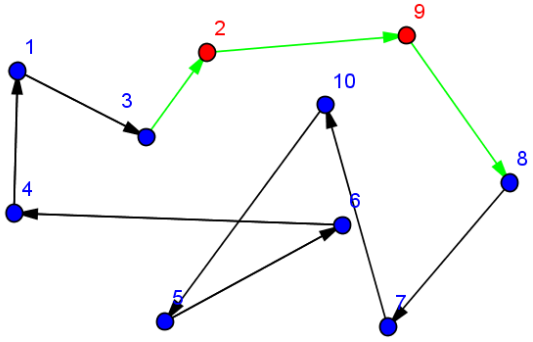
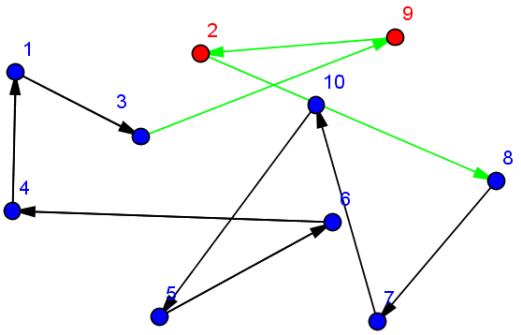


Figura 7 – Caminho $\bar{x'}$ obtido pela troca das cidades 2 e 9.



O efeito desta troca sobre a função objetivo pode ser calculado da seguinte forma: Dado um TTP1 com n cidades e m objetos, mas com matriz A nula e velocidade máxima $v_{máx}$, considere a troca da i-ésima cidade visitada pela cidade j-ésima cidade visitada (com $1 < i < j < n + 1$) numa dada solução $\bar{x}$, obtendo então uma nova solução $\bar{x'}$. Sem perda de generalidade (que ficará clara nos cálculos a seguir), suponha $i < j - 2$. Seja $d_{\bar{x}(k),\bar{x}(l)}$ a distância entre a k-ésima cidade visitada na solução $\bar{x}$ e a j-ésima cidade visitada também na solução $\bar{x}$. Portanto (lembrando que $\bar{x}(1) = \bar{x}(n + 1) = 1$),

$$f(\bar{x}) - f(\bar{x'}) = \sum_{k=1}^{n} t_{\bar{x}(k),\bar{x}(k+1)} - \sum_{l=1}^{n} t_{\bar{x'}(l),\bar{x'}(l+1)} = \\ = \frac{\sum_{k=1}^{n} d_{\bar{x}(k),\bar{x}(k+1)} - \sum_{l=1}^{n} d_{\bar{x'}(l),\bar{x'}(l+1)}}{v_{máx}} \tag{3.2}$$

e tendo em vista que todas as arestas em ambas as soluções são iguais a menos de quatro arestas em cada solução,

$$\begin{aligned} &d_{\bar{x}(i-1),\bar{x}(i)} + d_{\bar{x}(i),\bar{x}(i+1)} + d_{\bar{x}(j-1),\bar{x}(j)} + d_{\bar{x}(j),\bar{x}(j+1)} \\ &-d_{\bar{x'}(i-1),\bar{x'}(i)} - d_{\bar{x'}(i),\bar{x'}(i+1)} - d_{\bar{x'}(j-1),\bar{x'}(j)} - d_{\bar{x'}(j),\bar{x'}(j+1)} \end{aligned} \tag{3.3}$$

substituindo, deixando todos os termos denotados segundo a primeira solução:

$$\begin{aligned} &d_{\bar{x}(i-1),\bar{x}(i)} + d_{\bar{x}(i),\bar{x}(i+1)} + d_{\bar{x}(j-1),\bar{x}(j)} + d_{\bar{x}(j),\bar{x}(j+1)} \\ &-d_{\bar{x}(i-1),\bar{x}(j)} - d_{\bar{x}(j),\bar{x}(i+1)} - d_{\bar{x}(j-1),\bar{x}(i)} - d_{\bar{x}(i),\bar{x}(j+1)} \end{aligned} \tag{3.4}$$

Repare que se $j = i + 2$ então têm-se que $j - 1 = i + 1$ na Equação 3.4, enquanto que se $j = i + 1$ então $j - 1 = i$ (e como já dito $i + 1 = j$) na Equação 3.4, de modo que é possível fazer mais um passo de simplificações, mas a validade da equação continua independente do caso (tomando $d_{\bar{x}(k),\bar{x}(k)} = 0$).

É possível obter uma cota para a variação da função-objetivo após a troca de dois elementos de $\bar{x}$:

$$\begin{aligned} |G(\bar{x},(0,0,...,0)) - G(\bar{x'},(0,0,...,0))| &= |-R \times f(\bar{x}) - f(\bar{x'})| = \\ &|-R \times (\frac{d_{\bar{x}(i-1),\bar{x}(i)} + d_{\bar{x}(i),\bar{x}(i+1)} + d_{\bar{x}(j-1),\bar{x}(j)} + d_{\bar{x}(j),\bar{x}(j+1)}}{v_{máx}} \\ &\quad - \frac{d_{\bar{x}(i-1),\bar{x}(j)} + d_{\bar{x}(j),\bar{x}(i+1)} + d_{\bar{x}(j-1),\bar{x}(i)} + d_{\bar{x}(i),\bar{x}(j+1)}}{v_{máx}})| \leq \\ \leq \frac{R}{v_{máx}} \times (&|d_{\bar{x}(i-1),\bar{x}(i)}| + |d_{\bar{x}(i),\bar{x}(i+1)}| + |d_{\bar{x}(i+1),\bar{x}(j)}| + |d_{\bar{x}(j),\bar{x}(i-1)}| + \\ &|d_{\bar{x}(j-1),\bar{x}(j)}| + |d_{\bar{x}(j),\bar{x}(j+1)}| + |d_{\bar{x}(j+1),\bar{x}(i)}|) + |d_{\bar{x}(i),\bar{x}(j-1)}| \end{aligned} \tag{3.5}$$

Note que a desigualdade triangular aplicada permite cotar a função-objetivo em função do comprimento dos ciclos $\{i-1, i, i+1, j, i-1\}$ e $\{j-1, j, j+1, i, j-1\}$, de modo que o efeito sobre a função-objetivo causado pela troca da ordem de visita de duas cidades pode ser cotado de forma proporcional ao tamanho dos ciclos formados pelas cidades "adjacentes" (em ordem de visita) às cidades trocadas (inclusive). Este efeito pode ser visualizado pela Figura 8 e Figura 9 que mesclam a Figura 2 com a Figura 3 .

Assim, exceto pelo fato dos objetos serem desconsiderados, um TTP1 com cidades próximas (arestas de peso pequeno) terá a característica de ter o valor da função-objetivo de suas soluções menos afetada caso seja feita a troca da ordem de visita de duas cidades que

Figura 8 – Ciclo formado pelas cidades 2, 10 e suas adjacentes.

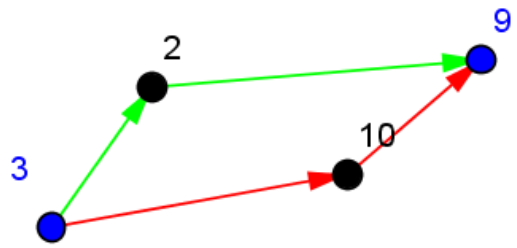
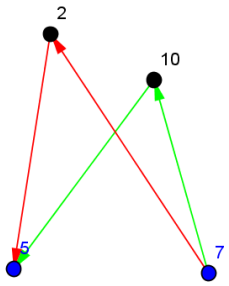


Figura 9 – Outro ciclo formado pelas cidades 2, 10 e suas adjacentes.



não sejam a cidade $\{1\}$. De fato, é possível obter uma melhor cota pela Equação 3.4. Esta outra cota pode ser melhor avaliada reorganizando esta equação obtendo a Equação 3.6.

$$\begin{aligned} |f(\bar{x}) - f(\bar{x'})| = \frac{R}{v_{máx}} \times |&d_{\bar{x}(i-1),\bar{x}(i)} - d_{\bar{x}(i-1),\bar{x}(j)} \\ &+d_{\bar{x}(i),\bar{x}(i+1)} - d_{\bar{x}(j),\bar{x}(i+1)} \\ &+d_{\bar{x}(j-1),\bar{x}(j)} - d_{\bar{x}(j-1),\bar{x}(i)} \\ &+d_{\bar{x}(j),\bar{x}(j+1)} - d_{\bar{x}(i),\bar{x}(j+1)}| \end{aligned} \tag{3.6}$$

Observa-se que se a aresta que liga uma das cidades adjacentes a $\bar{x}_i$ $(\bar{x}_j)$ tiver comprimento próximo ao da aresta que liga a mesma cidade a $\bar{x}_j$ $(\bar{x}_i)$ então o valor da função-objetivo será pouco alterado por este termo da Equação 3.6, e o mesmo vale para os outros três termos desta equação. Como um exemplo simples, basta imaginar que todas as cidades são equidistantes umas das outras, de modo que $d_{k,l} = a \in \mathbb{R}\ \forall k, l$.

O efeito da troca de posição de dois elementos de $\bar{x}$ quando há objetos a serem considerados será avaliado na próxima seção.

3.2 Efeito da troca de duas cidades em soluções do TTP1.

Suponha agora que a matriz A não é nula, ou seja, há a possibilidade de objetos serem coletados. Suponha então o mesmo problema anterior, mas agora com uma matriz A arbitrária. Seja uma solução $(\bar{x}, \bar{z})$ e suponha novamente a troca de dois elementos x_i e x_j como mostrado a seguir:

Suponha também que $\bar{z}$ e $\bar{z'}$ diferem apenas nos termos correspondentes às cidades $x_i = x'_j$ e $x_j = x'_i$. Nesta troca, deve-se considerar a troca de cidades nas quais objetos

$\bar{x} =$	x_1	x_2	...	x_{i-1}	x_i	x_{i+1}	...	x_{j-1}	x_j	x_{j+1}	...	x_{n-1}	x_n	x_1
$\bar{x'} =$	x_1	x_2	...	x_{i-1}	x_j	x_{i+1}	...	x_{j-1}	x_i	x_{j+1}	...	x_{n-1}	x_n	x_1

Tabela 6 – Troca de duas cidades em $\bar{x}$.

$\bar{z} =$	z_1	...	z_m
$\bar{z'} =$	z'_1	...	z'_m

Tabela 7 – $\bar{z}$ e $\bar{z'}$ correspondentes

foram recolhidos, pois não apenas serão trocadas as arestas, mas também será trocada a ordem de coleta de objetos, ocorrendo numa variação do peso da mochila durante o percurso do ladrão entre a visita de x_i e x_j. Denotando a velocidade v_O do ladrão dado que ele possui os k objetos do conjunto $I \supset O = \{I_{\alpha 1}, ..., I_{\alpha k}\}$, onde I é o conjunto de objetos disponíveis, tem-se que o tempo gasto em cada trecho do trajeto será (supondo $1 < i < j < n + 1$):

$$\begin{aligned} f(\bar{x}, \bar{z}) - f(\bar{x'}, \bar{z'}) = \sum_{k=1}^{n} t_{\bar{x}(k),\bar{x}(k+1)} - \sum_{l=1}^{n} t_{\bar{x'}(l),\bar{x'}(l+1)} = \\ \sum_{k=1}^{i-2} (t_{\bar{x}(k),\bar{x}(k+1)} - t_{\bar{x'}(k),\bar{x'}(k+1)}) \\ + t_{\bar{x}(i-1),\bar{x}(i)} + t_{\bar{x}(i),\bar{x}(i+1)} - t_{\bar{x'}(i-1),\bar{x'}(i)} - t_{\bar{x'}(i),\bar{x'}(i+1)} \\ + \sum_{k=i+1}^{j-2} (t_{\bar{x}(k),\bar{x}(k+1)} - t_{\bar{x'}(k),\bar{x'}(k+1)}) \\ + t_{\bar{x}(j-1),\bar{x}(j)} + t_{\bar{x}(j),\bar{x}(j+1)} - t_{\bar{x'}(j-1),\bar{x'}(j)} + t_{\bar{x'}(j),\bar{x}(j+1)} \\ + \sum_{k=j+1}^{n} (t_{\bar{x}(k),\bar{x}(k+1)} - t_{\bar{x'}(k),\bar{x'}(k+1)}) \end{aligned} \tag{3.7}$$

Esta equação pode ser simplificada ao se considerar que $t_{\bar{x}(k),\bar{x}(k+1)} = t_{\bar{x}(k),\bar{x'}(k+1)}$ se $k < i - 1$ pois nestes trechos os caminhos coincidem e para cada k a mochila conterá a

mesma coleção de objetos tanto em $\bar{x}$ quanto em $\bar{x'}$. Assim,

$$\begin{aligned} f(\bar{x},\bar{z}) - f(\bar{x'},\bar{z}) = \sum_{k=1}^{n} t_{\bar{x}(k),\bar{x}(k+1)} - \sum_{l=1}^{n} t_{\bar{x'}(l),\bar{x'}(l+1)} = \\ t_{\bar{x}(i-1),\bar{x}(i)} + t_{\bar{x}(i),\bar{x}(i+1)} - t_{\bar{x'}(i-1),\bar{x'}(i)} - t_{\bar{x'}(i),\bar{x'}(i+1)} \\ + t_{\bar{x}(j-1),\bar{x}(j)} + t_{\bar{x}(j),\bar{x}(j+1)} - t_{\bar{x'}(j-1),\bar{x'}(j)} + t_{\bar{x'}(j),\bar{x'}(j+1)} \\ + \sum_{k=i+1}^{j-2} (t_{\bar{x}(k),\bar{x}(k+1)} - t_{\bar{x'}(k),\bar{x'}(k+1)}) \\ + \sum_{k=j+1}^{n} (t_{\bar{x}(k),\bar{x}(k+1)} - t_{\bar{x'}(k),\bar{x'}(k+1)}) \end{aligned} \tag{3.8}$$

Denotando $O_l \subset I$ como o conjunto de objetos contidos na mochila logo após o ladrão deixar a cidade $\bar{x}(l)$ no percurso $\bar{x}$, ou melhor, $O_l = \{k \in \{1,...,m\} | 0 < \bar{z}(k) \leq \bar{x}(l)\}$. Análogamente denote $O'_l \subset I$ para o percurso $\bar{x'}$ e escrevendo $d_{\bar{x}(a),\bar{x}(b)} = \sum l = ab - 1 d_{\bar{x}(l),\bar{x}(l+1)}$ (a<b) a Equação 3.8 pode ser reescrita:

$$\begin{aligned} \Delta((\bar{x},\bar{z}),(\bar{x'},\bar{z'})) = f(\bar{x},\bar{z}) - f(\bar{x'},\bar{z}) = \sum_{k=1}^{n} t_{\bar{x}(k),\bar{x}(k+1)} - \sum_{l=1}^{n} t_{\bar{x'}(l),\bar{x'}(l+1)} = \\ \frac{d_{\bar{x}(i-1),\bar{x}(i)}}{v_{O_{i-1}}} + \frac{d_{\bar{x}(i),\bar{x}(i+1)}}{v_{O_i}} - \frac{d_{\bar{x'}(i-1),\bar{x'}(i)}}{v_{O'_{i-1}}} + \frac{d_{\bar{x'}(i),\bar{x'}(i+1)}}{v_{O'_i}} \\ + \frac{d_{\bar{x}(j-1),\bar{x}(j)}}{v_{O_{j-1}}} + \frac{d_{\bar{x}(i-1),\bar{x}(j)}}{v_{O_{j+1}}} - \frac{d_{\bar{x'}(j-1),\bar{x'}(j)}}{v_{O'_{j-1}}} + \frac{d_{\bar{x'}(j),\bar{x'}(j+1)}}{v_{O'_j}} \\ + t_{\bar{x}(i+1),\bar{x}(j-1)} - t_{\bar{x'}(i+1),\bar{x'}(j-1)} \\ + t_{\bar{x}(j+1),\bar{x}(n+1)} - t_{\bar{x}(j+1),\bar{x}(n+1)} \end{aligned} \tag{3.9}$$

Lembrando que os termos das duas últimas linhas da Equação 3.9 diferem apenas na velocidade mas não na distância percorrida. Como a velocidade depende linearmente do peso da mochila, então $v_{O_k} = v_{O'_k} + c_l$ onde $i+1 \leq k \leq j+1$ e $l \in \{i,j\}$ com c_l sendo a diferença dos pesos da mochila na l-ésima cidade em cada uma das soluções $(\bar{x},\bar{z})$ $(\bar{x'},\bar{z'})$. Assim é possível reduzir ainda mais a Equação 3.9 sabendo-se que $v_{O_{i-1}} = v_{O'_{i-1}}$ e escrevendo $v_{O_i} = v_{O'_i} + c_i$ tem-se $v_{O_{j-1}} = v_{O'_{j-1}} + c_i$ além de ser possível escrever $v_{O_j} = v_{O'_j} + c_j$, mas esta reescrita não é necessária para o objetivo deste texto e será deixada apenas como observação.

Apesar da Equação 3.9 não explicitar os efeitos de trocas mais complexas, ou mesmo a diferença entre duas soluções arbitrárias com relação à função-objetivo, ela é

o suficiente para explicitar a variação sofrida entre soluções nos operadores da próxima seção.

3.3 Operadores genéticos.

Na Equação 3.9 foi avaliado o efeito da troca de duas cidades de uma dada solução supondo que os objetos retirados na antiga solução e na nova diferem apenas nos termos relacionados às cidades trocadas. Tendo isto em mãos, definimos um operador que dirá *quais soluções (se existirem) se encontram entre duas soluções dadas*. Este operador é explicitado no algoritmo 2 abaixo.

Algoritmo 2: Encontrar soluções intermediárias.

Entrada: $(\bar{x}, \bar{z})$, $(\bar{x'}, \bar{z'})$
1 $(\bar{x}_1, \bar{z}_1), ..., (\bar{x}_k, \bar{z}_k)$ $t \leftarrow n$
2 $i \leftarrow 1$
3 $(\bar{x}_s, \bar{z}_s) \leftarrow (\bar{x}, \bar{z})$
4 $(\bar{x}_s, \bar{z}_s) \neq (\bar{x'}, \bar{z'})$ $\bar{x}_s(t) \neq \bar{x'}(t)$ **Encontre o k tal que** $\bar{x}_s(k) = \bar{x'}(t)$
5 $\bar{x}_s(k) \leftarrow \bar{x}_s(t)$
6 $\bar{x}_s(t) \leftarrow \bar{x'}(t)$
7 **Encontre o conjunto** $E = \{e | \bar{z}_s(e) = \bar{x}_s(t)\}$
8 **Faça** $\bar{z}_s(e) \leftarrow 0$ $\forall e \in E$
9 **Encontre o conjunto** $E' = \{e' | \bar{z'}(e') = \bar{x}_s(t)\}$
10 **Faça** $\bar{z}_s(e') \leftarrow \bar{x}_s(t)$ $\forall e \in E'$
11 $(\bar{x}_s, \bar{z}_s) \neq (\bar{x'}, \bar{z'})$ $(\bar{x}_i, \bar{z}_i) \leftarrow (\bar{x}_s, \bar{z}_s)$
12 $i \leftarrow i + 1$ $t \leftarrow t - 1$

Repare que neste algoritmo ainda é feito o teste para verificar se a solução $(\bar{x}_s, \bar{z}_s)$ é diferente de $(\bar{x'}, \bar{z'})$ antes de ser considerada uma das soluções intermediárias, pois existem casos em que após uma simples troca é possível obter uma solução a partir de outra (como exemplificado na Equação 2.4.1 do Capítulo 2). Além disso, este algoritmo tem retorno nulo caso as duas entradas sejam iguais, como é de se esperar. Este algoritmo possui duas propriedades muito importantes:

- Este operador não utiliza qualquer parâmetro do TTP1 do qual suas entradas estão relacionadas. Além das soluções dadas na entrada do algoritmo, o único parâmetro intrínseco ao TTP1 que foi utilizado é o número de cidades (n) que pode ser obtido pelo comprimento dos vetores $\bar{x}$ ou $\bar{x'}$ dados.

- Todos os valores de $\bar{x}_i$ retornados por este algoritmo serão permutações das cidades de modo a manter o formato $\bar{x}_i = \{1, \bar{x}_i(1), ..., \bar{x}_i(n), 1\}$ onde $\bar{x}_i(j) \neq \bar{x}_(k)$ se $j \neq k$ e $\bar{x}_i(j) \in \{2, 3, ..., n\}$ $\forall j$. Logo, todo $\bar{x}_i$ será factível se o grafo G do TTP1 tratado for um grafo completo.

- Todos os valores de $\bar{z}_i$ retornados por este algoritmo serão factíveis a menos da restrição de peso da mochila, i.e, se $\bar{z}_i(k) \neq 0$ então $a_{k,\bar{z}_i(k)} = 1$ por construção. Porém, nada garante que o peso dos objetos selecionados em $\bar{z}_i$ não ultrapasse a capacidade máxima da mochila.

Assim, supondo um grafo completo e uma mochila que suporte qualquer combinação de objetos (ou melhor, uma mochila que suporte todos os objetos) então este operador sempre obterá (se retornar alguma solução) soluções factíveis para o TTP1 dado sem precisar fazer qualquer consulta à matriz A de disponibilidade de objetos, ou consultar I para avaliar se os pesos dos objetos ultrapassam a capacidade de carga W. A seguir é mostrado um exemplo de aplicação deste operador.

Suponha um TTP1 com 7 cidades e 5 objetos. Tome $(\bar{x}, \bar{z})$, $(\bar{x'}, \bar{z'})$ como mostrado na algoritmo 12. O fluxo das operações feitas no algoritmo é mostrado segundo as setas indicam, de modo a avaliar se a solução $\bar{x}_s$ já coincide com $\bar{x'}$, fazer a troca, mudar $\bar{z}_s$ em duas etapas, e por fim avaliar se $(\bar{x}_s, \bar{z}_s)$ já se igualou a $(\bar{x'}, \bar{z'})$ antes de obter uma solução $(\bar{x}_i, \bar{z}_i)$ intermediária. Estes passos são repetidos até serem encontradas três soluções intermediárias, indicando a ordem das operações por setas, e sublinhando em vermelho as cidades trocadas em cada passo, e em azul os objetos cuja retirada é modificada.

Note que neste exemplo, a diferença entre a solução mais "próxima" de $(\bar{x'}, \bar{z'})$ (a saber: $(\bar{x}_3, \bar{z}_3)$) é que o quinto objeto é retirado na cidade 4 na nova solução, enquanto este objeto não é retirado na solução final $(\bar{x'}, \bar{z'})$. Por construção do algoritmo que pára se a primeira parte da solução nova (o trajeto $\bar{x}_s$) é igual ao da solução $(\bar{x'}, \bar{z'})$ então a solução obtida que é mais próxima de $(\bar{x'}, \bar{z'})$ irá diferir desta apenas em sua segunda componente $(\bar{z'})$). Ou seja, basta fazer uma operação de "correção do $\bar{z}$" (semelhante à do algoritmo) que anula os $\bar{z}_s(i))$ que diferem dos $(\bar{z'}(i))$ e em seguida os corrige de forma a tornar $\bar{z}_s = \bar{z'}$).

Assim definimos este operador como o operador **inter**, onde $\textbf{inter}((\bar{x}, \bar{z}), (\bar{x'}, \bar{z'})) = \{(\bar{x}_1, \bar{z}_1), ..., (\bar{x}_k, \bar{z}_k)\}$ é o operador que encontra as k soluções que se encontram entre $(\bar{x}, \bar{z})$

$\bar{x} =$	1	2	**3**	4	5	6	**7**	1		$\bar{z} =$	7	3	0	5	4
↓															
$\bar{x}_s =$	1	2	**7**	4	5	6	**3**	1	→	$\bar{z}_s =$	7	**0**	0	5	4
										↓					
										$\bar{z}_s =$	**3**	0	0	5	4
$\bar{x}_1 =$	1	2	**7**	4	**5**	6	3	1		$\bar{z}_1 =$	3	0	0	5	4
↓															
$\bar{x}_s =$	1	2	**5**	4	**7**	6	3	1	→	$\bar{z}_s =$	3	0	0	5	4
										↓					
										$\bar{z}_s =$	3	0	**7**	5	4
$\bar{x}_2 =$	1	**2**	**5**	4	7	6	3	1		$\bar{z}_2 =$	3	0	7	5	4
↓															
$\bar{x}_s =$	1	**5**	**2**	4	7	6	3	1	→	$\bar{z}_s =$	3	0	7	5	4
										↓					
										$\bar{z}_s =$	3	0	7	5	4
$\bar{x}_3 =$	1	5	2	4	7	6	3	1		$\bar{z}_3 =$	3	0	7	5	**4**
↓										↓					
$\bar{x}' =$	1	5	2	4	7	6	3	1		$\bar{z}' =$	3	0	7	5	0

Tabela 8 – Encontrando as soluções que se encontram "entre" $(\bar{x}, \bar{z})$ e $(\bar{x}', \bar{z}')$.

e $(\bar{x}', \bar{z}')$ (segundo a definição de "soluções intermediárias" dada) com[2] $0 \leq k \leq n-2$.

Logo, calculando **inter**$((\bar{x}, \bar{z}), (\bar{x}', \bar{z}'))$, se sua saída não for nula, basta escolher um dos elementos encontrados como sendo o filho do cruzamento destas duas soluções. Além disso, supondo que este operador encontrou k soluções, escolhendo uma solução $(\bar{x}_i, \bar{z}_i)$ com i pequeno, então esta solução será mais "parecida com $(\bar{x}, \bar{z})$", enquanto que se i for próximo de k, esta solução será mais "parecida com $(\bar{x}', \bar{z}')$". Este critério, apesar de um tanto informal, pode ser utilizado tendo em vista a quantidade de arestas iguais entre uma solução pai e a filha, bem como os objetos recolhidos e onde eles foram recolhidos. Assumindo o operador **inter**, denota-se a semelhança entre um progenitor e seu filho por esta medida.

Quanto a um possível operador de mutação, a troca de duas cidades em $\bar{x}$ será utilizada, de modo a sortear genes deste cromossomo a serem mutados (e portanto, trocados de posição) segundo uma taxa de mutação escolhida pelo programador. Porém $\bar{z}$ deve ter um operador de mutação diferente, de modo que a mutação de uma solução factível permaneça factível, e para isso é necessário consultar a matriz A do TTP1 tratado. Assim, dada uma taxa de mutação escolhida, sorteia-se genes de $\bar{z}$ a serem mutados, de modo

[2] Suponha que (no algoritmo 2) foram feitas trocas em $\bar{x}$ para todo valor de t até t=3. Então após n-3 trocas, o próximo elemento a ser trocado será o terceiro elemeto de $\bar{x}$ que necessariamente deve ser trocado com o segundo elemento (já que todos os demais já estão em seus devidos lugares graças às iterações anteriores) totalizando n-2 trocas.

que se o i-ésimo gene foi sortedado, deve-se escolher um j tal que $a_{i,j} = 1$, e em seguida substituir este gene fazendo $\bar{z}(i) = j$. Outras possibilidades que permitem a mutação de $\bar{z}$ sem a consulta de A (que pode ser demorada dependendo das dimensões de A) poderia ser feita via consulta de outras soluções da população de soluções factíveis geradas inicialmente no algoritmo genético (que utilizou A), de modo a permitir obter algumas informações sobre quais cidades contêm cada objeto. Outra opção (mas que dificilmente irá gerar boas soluções) é anular os genes sorteados para mutação, pois $\{0\}$ é um valor que será sempre válido em qualquer gene, além de garantir que a solução mutada será factível desde que a solução antes de sofrer mutação também seja factível. Nesta última opção de operador de mutação, uma taxa de mutação alta pode causar a anulação dos $\bar{z}$ da população de soluções com o passar das gerações, reduzindo o TTP1 a um TSP pois o operador de cruzamento irá manter estes valores nulos de $\bar{z}$.

No próximo capítulo será discutido especificamente o algoritmo genético estudado, detalhando os operadores utilizados, e mostrando os resultados obtidos por este algoritmo que servirão de base de comparação às soluções obtidas no segundo algoritmo dado no Capítulo 5.

4 Metodologia - Algoritmo Genético

Neste capítulo é feita a descrição do algoritmo genético que será utilizado como base para comparação do segundo algoritmo que é descrito Capítulo 5.

Na seção 4.1 é feita a descrição completa do algoritmo genético de modo a apresentar seus operadores e parâmetros.

Na seção 4.2 é mostrado um exemplo que foi processado pelo algoritmo, avaliando a evolução da população de soluções em cada geração.

4.1 Descrição do algoritmo genético

Como mencionado na seção 2.4 do Capítulo 2 a ausência de estudos que utilizem a estrutura do TTP1 torna justificável a utilização de um algoritmo genético como uma primeira tentativa de resolvê-lo. Neste estudo foi utilizado o algoritmo 3 abaixo:

Algoritmo 3: Algoritmo genético utilizado

Entrada: TTP1, parâmetros do algoritmo genético

1 Populacao de soluções $t \leftarrow 0$
2 **Gere população inicial** P_t **com** $\mathbf{N_g}$ **indivíduos**
3 **Avalie** P_t
4 Critérios de parada não forem satisfeitos **Selecione** $\mathbf{N_g}$ **pares** de indivíduos progenitores
5 **Cruze** os progenitores (com **mutação**), obtendo uma nova população F_t de $\mathbf{N_g}$ filhos
6 **Avalie** F_t
7 **Selecione** os indivíduos de $P_t \bigcup F_t$, que farão a nova geração P_{t+1}
8 **Guarde** em P_{t+1} as aptidões dos indivíduos selecionados
9 $t \leftarrow t + 1$

Na linha (2) é gerada uma população inicial P_t $(t = 0)$ de tamanho $\mathrm{N_g}$ que será avaliada logo abaixo, na linha (3). Na (4) são iniciadas as iterações que farão a evolução da população obtendo novas gerações de indivíduos. O critério de parada do algoritmo fica a critério do programador, podendo ser pela variância da função-objetivo dos indivíduos da geração t (se a variância for menor que um valor var_{min} escolhido), pela quantidade de gerações obtidas (se $t > t_{max}$) e etc.

A evolução da população inicial é iniciada, e na linha (4) são selecionados $2N_g$ indivíduos progenitores[1]. O critério de seleção fica a cargo do programador, sendo tipicamente algum critério probabilístico que tenha a tendência de escolher os progenitores mais aptos (maior valor da função-objetivo) sem descartar completamente os indivíduos menos aptos (com o fim de manter a diversidade da população). O critério de escolha utilizado foi o **torneio binário**[2]. Assim, são feitos $2N_g$ torneios, e os $2N_g$ vencedores são os progenitores selecionados para recombinação.

Na linha (5) é feito o cruzamento (com mutação) dos $2N_g$ progenitores (cruzando o i-ésimo indivíduo com o (i+N_g)-ésimo indivíduo) obtendo N_g filhos, obtendo uma nova população chamada de F_t. O cruzamento utilizado é o algoritmo 2 descrito no Capítulo 3 de modo que o indivíduo obtido pela recombinação de dois progenitores será um dos indivíduos obtidos pelo operador **inter**() se sua saída for não-nula. Caso **inter**() tenha saída nula, o indivíduo gerado pelo cruzamento será uma cópia (escolhida aleatoriamente) de um de seus progenitores. A nova população é então avaliada na linha (6).

Na linha (7) é obtida a nova geração P_{t+1} de indivíduos a partir de P_t e F_t. Para obter esta nova geração são selecionados N_g indivíduos dentre os $2N_g$ indivíduos de $P_t \bigcup F_t$ utilizando um critério de seleção escolhido pelo programador. Neste estudo foi escolhida a seleção por **roleta de N_g agulhas**[3] (ou SUS - Stochastic Universal Sampling) de modo que todo indivíduo pode ser escolhido, dando preferência para os mais aptos mas sem suprimir os menos aptos (e manter a diversidade).

Finalmente, a nova população P_{t+1} é avaliada na linha (8) (pois a aptidão dos indivíduos da nova geração pode ser utilizada nos critérios de parada do algoritmo) e na linha (9) é atualizado o índice da geração de indivíduos (que também pode ser um dos critérios de parada).

Na próxima seção será mostrado um exemplo de aplicação do algoritmo genético.

---

1 Repare que um mesmo indivíduo pode ser escolhido como progenitor para mais de um cruzamento.

2 Neste torneio são escolhidos dois indivíduos aleatoriamente, e em seguida é feito um "torneio" onde o mais apto será o vencedor. O indivíduo vencedor deste torneio será o indivíduo selecionado pelo torneio binário.

3 Como é possível que indivíduos possuam aptidão negativa foi utilizado um sistema de ranking, onde o pior indivíduo de uma população recebeu ranking 1, o segundo pior recebeu ranking 2 e assim por diante. O ranking de cada indivíduo determinará a chance de ser selecionado pela roleta.

4.2 Exemplo de aplicação do algoritmo

Tome um TTP1 com os seguintes parâmetros:

- n = 100 cidades
- m = 8 objetos diferentes
- W = 6 (peso máximo da mochila)
- R = \$1 de aluguel por unidade de tempo
- $v_{máx} = 1$ e $v_{mín} = 0.1$
- Pesos dos objetos: [1 1 2 3 2 2 3 1]
- Valores dos objetos: [40 60 60 70 40 30 30 15]

A matriz de distâncias e a matriz de disponibilidade dos objetos será ocultada devido ao grande tamanho destas, mas ressalta-se que a distribuição dos objetos foi probabilística, de modo que cada objeto possui uma probabilidade (escolhida pelo programador) de ser colocado em cada cidade durante a criação do problema. Quanto às distâncias entre as cidades, foi criado um problema onde cada cidade se localiza no plano, e cada cidade está conectada às outras por uma linha reta (equivalente a um grafo completo cujas arestas são dadas pela distância euclidiana de seus vértices). A Figura 10 abaixo ilustra o posicionamento de cada cidade. A cidade **1** se encontra na posição (0,0) (círculo vermelho) enquanto as demais se encontram aglomeradas em regiões diferentes em cada quadrante. Cada aglomerado possui 25 cidades exceto pelo posicionado no primeiro quadrante que possui 24 cidades (totalizando 100 cidades ao contabilizar a cidade 1). Cada um destes aglomerados será denotado por **cluster** e serão importantes para a abordagem estudada no Capítulo 5.

Foi executado o algoritmo 3 com os seguintes parâmetros:

- $N_g = 100$
- Seleção de progenitores: torneio binário (N_g torneios por seleção)

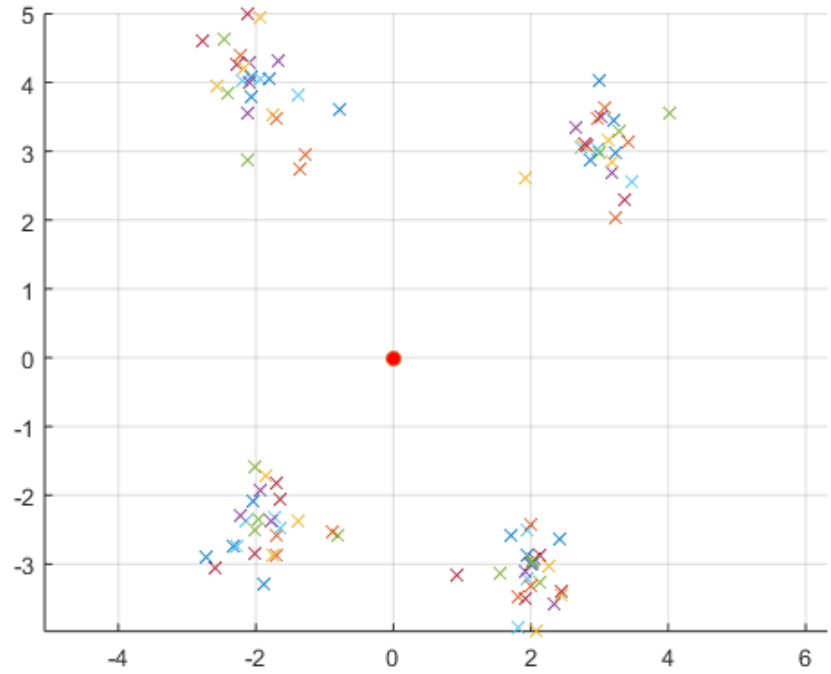


Figura 10 – Posição cartesiana de cada cidade do TTP1 dado.

- Cruzamento utilizando o operador **inter()** e selecionando, se existir, o melhor indivíduo intermediário aos progenitores. Caso não exista, é selecionado um progenitor aleatório e uma cópia deste indivíduo será o resultado do cruzamento.

- Mutação do primeiro cromossomo $\bar{x}$: troca de genes adjacentes

- Mutação do segundo cromossomo $\bar{z}$: substituição do gene por outro possível (lendo a matriz A de disponibilidade de objetos)

- Chance de mutação por gene: 5% (para ambos os cromossomos)

- Seleção dos componentes da próxima geração: roleta de N_g agulhas

- Critério de parada 1: Variância da aptidão dos indivíduos de uma geração menor que 3

- Critério de parada 2: Numéro de gerações produzidas (iterações do algoritmo) passar de 100

Ressalta-se que neste algoritmo não é verificado se as soluções obtidas são factíveis com relação à restrição de peso $W_c < W$ de modo a agilizar o algoritmo sem que tenha que fazer muitas tentativas mal-sucedidas nos operadores de cruzamento, ou mesmo muitas consultas a grandes matrizes como A. Para contornar este problema, de modo que sejam obtidas (pelo menos em grande parte) soluções factíveis foi utilizado o método de penalização ([2]) de modo que soluções infactíveis não sejam descartadas durante o processo evolutivo, mas penalizadas em suas aptidões. Isto foi feito modificando o modo como a aptidão de um indivíduo é calculada.

A forma de se calcular a velocidade do caixeiro caso o seu peso seja menor ou igual ao peso máximo suportado pela mochila permanece o mesmo, mas caso seu peso ultrapasse o peso máximo da mochila sua velocidade será calculada da seguinte forma:

$$v_c = \frac{v_{min}}{W_c^2}, W_c > W \tag{4.1}$$

No exemplo mostrado, $W_c = 3$ logo por menor que seja a diferença entre o peso corrente e o peso máximo, a velocidade que seria v_{min} se $W_c = W$ passa a ser pelo menos $\frac{vmin}{9}$ se $W_c = W + \epsilon$ por menor que seja ϵ, e quanto maior for ϵ menor ainda será a velocidade do ladrão (graças ao termo quadrático). Com esta grande penalização na velocidade do caixeiro quando uma solução for infactível, o algoritmo genético passará a ter uma tendência a escolher soluções factíveis para comporem as próximas iterações do algoritmo. Caso o algoritmo continue selecionando soluções infactíveis, basta o programador aumentar a penalidade para soluções infactíveis, encontrando os α e β satisfatórios na Equação 4.2, lembrando de não impor uma penalidade muito grande a soluções infactíveis caso seja desejável (e em geral é) que soluções infactíveis sejam utilizadas nas gerações iniciais de modo a no final do algoritmo obter uma população factível e com uma aptidão desejável. Outras formas de penalização também podem ser utilizadas, mas a escolhida neste estudo foi a forma dada na Equação 4.2, e neste exemplo em particular foi tomado $\alpha = 1$ e $\beta = 2$.

$$v_c = \frac{v_{min}}{\alpha \times W_c^{\beta}}, W_c > W \tag{4.2}$$

Com estes parâmetros, foram executadas 10 iterações do algoritmo obtendo 10 históricos diferentes de evolução das populações de soluções, de modo que algumas

alcançaram o critério de parada antes das outras, mas de modo geral os valores médios das funções-objetivo da população final de cada iteração não variou muito. A Figura 11 mostra a evolução dos indivíduos de cada população.

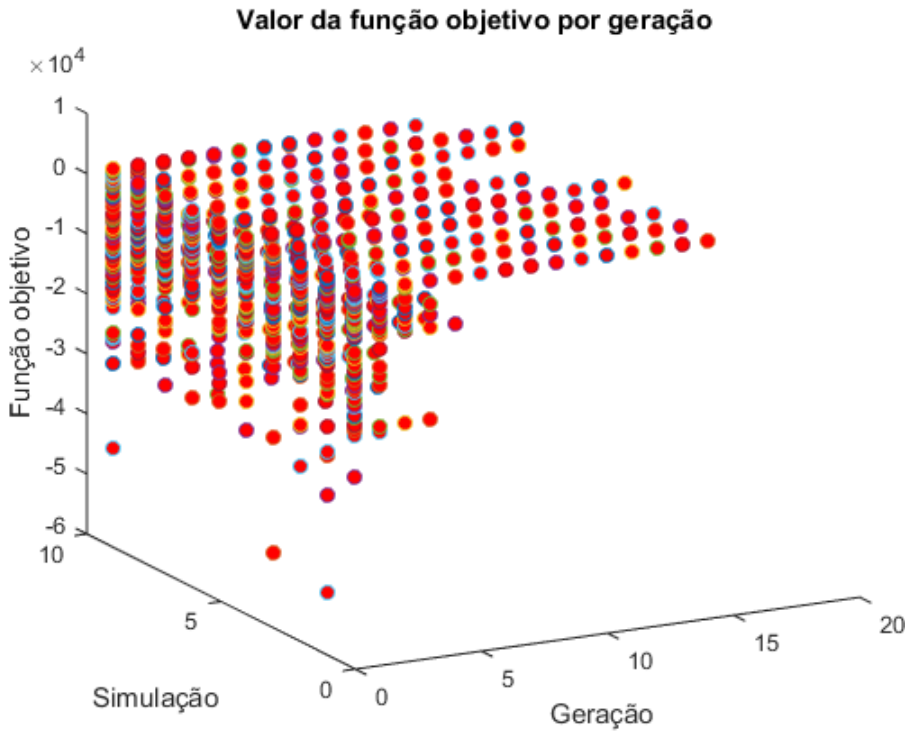


Figura 11 – Aptidão dos indivíduos de cada simulação do algoritmo genético.

A Figura 13 mostrada a evolução da população de apenas uma das simulações.

Repare que inicialmente a função-objetivo da primeira geração (que foi criada aleatoriamente) possui muitos indivíduos com valores muito negativos. Isso decorre do fato de esta primeira população ser gerada aleatoriamente, permitindo que muitos indivíduos sejam infactíveis e tenham uma grande penalidade em suas aptidões. Logo na segunda geração já são obtidos indivíduos com aptidão positiva devido à seleção feita durante a recombinação.

Na Figura 14 é mostrado o tempo gasto para gerar cada nova população do algoritmo nesta simulação, de forma cumulativa. O computador utilizado para a execução do algoritmo possui um processador Intel(R) Core(TM) i3-4005U de 1.70 GHz, 4GB de RAM e Windows 10 Home versão 1511. No caso da primeira similação foram necessários 126.95 segundos para um total de 15 gerações até que os critérios de parada sejam satisfeitos

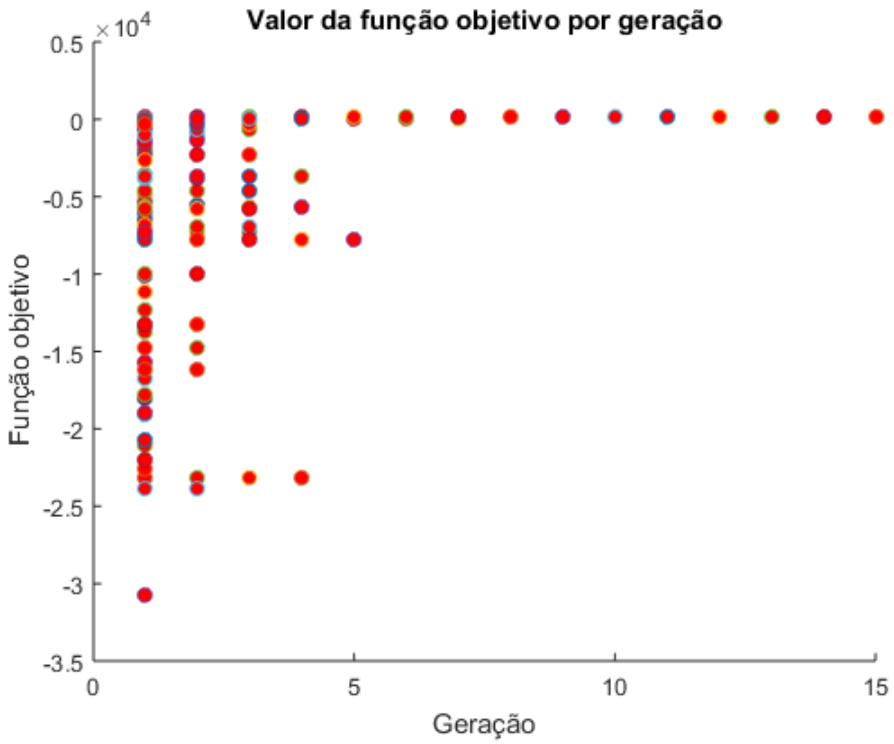


Figura 12 – Aptidão dos indivíduos da primeira simulação do algoritmo genético.

(neste caso o algoritmo parou de ser executado devido à variância da aptidão da geração 15 ser menor que 3). Repare que a partir da geração 8 há uma redução nos incrementos de tempo da figura, indicando que cada nova geração a partir desta passaram a demorar menos para serem geradas. Esta variação nos incrementos pode ser devido ao fato de o algoritmo ter encontrado um mínimo local para a função-objetivo e passar a fazer buscas dentro deste mínimo, onde as soluções não diferem muito umas das outras e portanto o cálculo do cruzamento é reduzido (por existirem menos soluções intermediárias entre duas soluções quaisquer). Foram utilizadas as funções **tic** e **toc** do MATLAB para a medição de tempo gasto pelo algoritmo.

Como pode ser visto na Figura 13 a população final possui aptidões próximas entre seus indivíduos. De fato, o valor médio das funções-objetivo da população final é de 157.59 com variância de 2.98 enquanto as demais simulações obtiveram valores semelhantes, ilustrados na Figura 11.

Outros tipos de operadores e parâmetros podem ser escolhidos obtendo resultados diferentes de convergência e valores finais das funções-objetivo dos indivíduos, mas neste

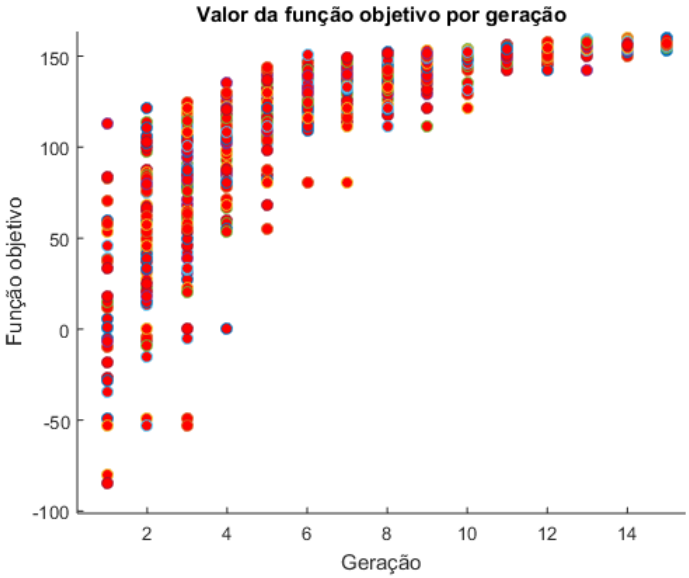


Figura 13 – Convergência da aptidão dos indivíduos da primeira simulação.

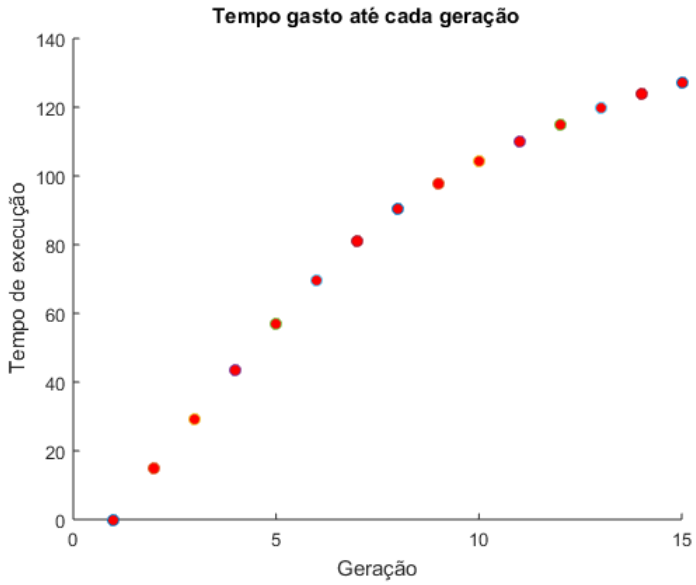


Figura 14 – Tempo gasto para executar o algoritmo.

estudo foram utilizados os operadores e parâmetros mencionados acima. No próximo capítulo é mostrada uma nova abordagem do problema que busca explorar as propriedades de uma classe específica deste problema.

5 Metodologia - Abordagem via *Clusters*

Neste capítulo será exposta a principal contribuição deste estudo que visa explorar características existentes em TTP1s específicos.

Na seção 5.1 é dada a principal motivação para a abordagem via clusters (que serão definidos na seção seguinte). São mostradas (de forma empírica) carcterísticas presentes em classes específicas do problema estudado fazendo referência ao exemplo estudado no capítulo anterior.

Na seção 5.2 é feita a definição de *clusters* neste estudo e é mostrada sua principal aplicação na resolução de casos específicos do TTP1. Esta propriedade é então aplicada ao exemplo trabalhado de modo a comparar os resultados obtidos com os do capítulo anterior.

No capítulo seguinte serão feitas simulações comparando o algoritmo genético do Capítulo 4 com o algoritmo construído neste capítulo de modo a avaliar até que ponto a abordagem por clusters se mostra eficaz.

5.1 Motivação

No Capítulo 4 foi dado um exemplo no qual a posição das cidades do problema não é completamente arbitrária. Na Figura 10 algumas cidades se encontram aglomeradas nos quatro quadrantes do plano, de modo que o aglomerado do primeiro quadrante possui 24 cidades enquanto os demais possuem 25 cidades e a cidade {1} se encontra na posição (0,0).

Estes aglomerados foram gerados propositalmente, de modo que as cidades {2} a {25} foram colocadas em posições aleatórias dentro do círculo de raio $\frac{\pi}{2}$ com centro na posição (3,3). As cidades {26} a {50}, {51} a {75}, e {76} a {100}, foram posicionadas aleatoriamente dentro dos círculos de mesmo raio com centros em (2,-3), (-2,4) e (-2,-2.5) respectivamente. Como dito anteriormente, a cidade {1} se encontra em (0,0). Por enquanto chamaremos o "aglomerado" de cidades posicionadas no i-ésimo quadrante como *aglomerado i*.

Além disto existem 8 objetos diferentes distribuídos nestas 100 cidades. A matriz de disponibilidades dos objetos é mostrada em quatro partes pelas quatro tabelas abaixo. Em cada tabela são destacadas as linhas onde aparecem 1's na matriz, de modo a mostrar quais objetos estão disponíveis em quais aglomerados. A Tabela 13 é uma versão simplificada da tabela de disponibilidades, mostrando quais objetos estão disponíveis em cada aglomerado.

0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0	1
0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabela 9 – Matriz de disponibilidade: colunas 1 a 25

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabela 10 – Matriz de disponibilidade: colunas 26 a 50

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabela 11 – Matriz de disponibilidade: colunas 51 a 75

Note que a primeira coluna da matriz de disponibilidades é nula, indicando que realmente não há objetos na cidade $\{1\}$. Além disso os objetos distribuídos se encontram concentrados nos aglomerados, como pode ser visto pela matriz de disponibilidade, além da Tabela 13 que mostra esta propriedade explicitamente.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabela 12 – Matriz de disponibilidade: colunas 76 a 100

Tabela 13 – Disponibilidade de objetos por aglomerado

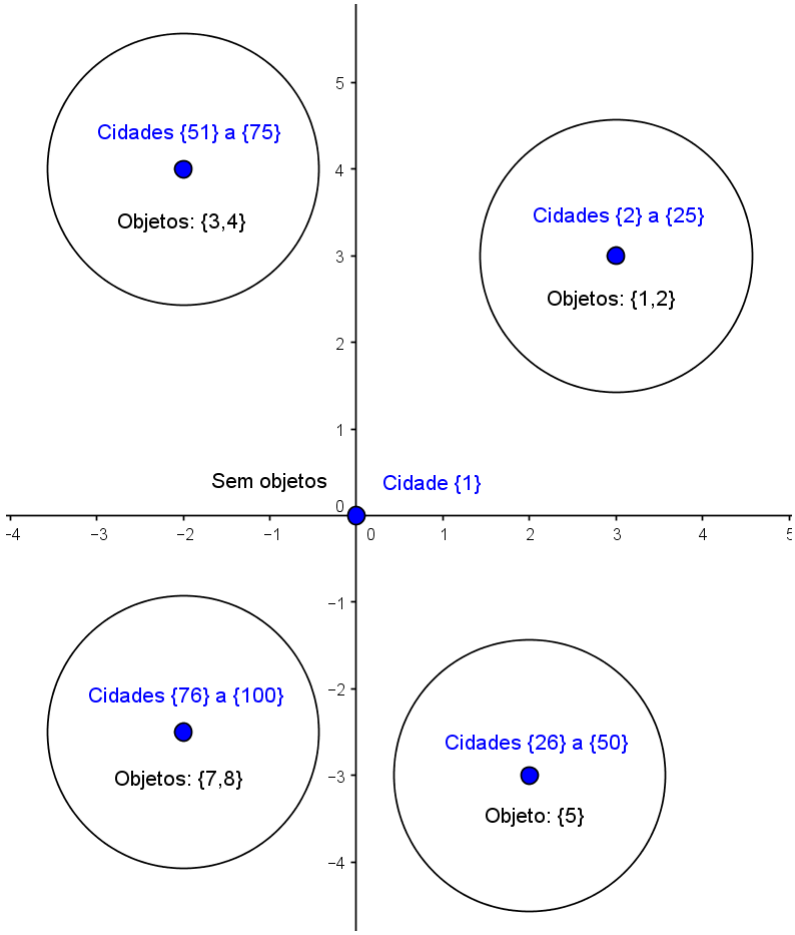
Aglomerado	Objetos
1	1,2
2	3,4
3	7,8
4	5

Quando um grupo de cidades se encontra espacialmente próximo (arestas de pequeno tamanho interligando cada vértice correspondente à estas cidades) e estas possuem os mesmos objetos disponíveis, é razoável imaginá-las como se fossem uma única cidade com tais objetos, como é mostrado na Figura 15.

Como exemplo, considere o aglomerado 1 composto pelas cidades $\{2, 3, ..., 25\}$. Este aglomerado de cidades está contido no círculo de raio $\frac{\pi}{2}$ centrado em $(3, 3)$ de modo que algumas cidades possuem o objeto $\{1\}$, enquanto outras possuem o objeto $\{2\}$ (a cidade $\{23\}$ possui ambos) como pode ser visto na Tabela 9. Assim, este aglomerado de cidades pode ser simplificado como uma única "grande cidade" a ser visitada, de modo que se o ladrão decidir percorrer alguma das cidades deste aglomerado, é plausível imaginar que ele queira aproveitar a oportunidade e já visitar as demais cidades da região, pois pode ser ineficiente sair desta região para visitar outras cidades para, em seguida, voltar para esta região e visitar as cidades faltantes. A grande motivação para tal está no fato de que os objetos $\{1\}$ e $\{2\}$ somente estão disponíveis neste aglomerado, logo, caso o ladrão decida coletar algum destes objetos então ele deve necessariamente entrar nesta região para efetuar a coleta.

Estes argumentos empíricos são a principal motivação para a definição de um novo objeto matemático que será chamado de *cluster* e será definido na próxima seção.

Figura 15 – Simplificação do problema dado.



5.2 Clusters - Definição e aplicação

Dado um TTP1 com um conjunto $C = \{1, ..., n\}$ de cidades e um conjunto $I = \{1, ..., m\}$ de itens e uma matriz de disponibilidade de objetos A. Um **cluster** $Cl_\gamma = \{J, O\}$ é definido como um conjunto de cidades $C \supset J = \{c_1, ..., c_k\}$ e um conjunto de objetos $I \supset O = \{o_1, ..., o_l\}$ tais que:

- $d_{c_i,c_j} \leq \gamma \ \forall i, j \in \{1, ..., k\}$
- Se $x \in C$ é uma cidade tal que o objeto o_l está disponível em x (ou melhor, $A_{o_l,x} = 1$) então $x \in J$
- Se $Cl'_{\gamma'} = \{J', O'\}$ é outro cluster, então $J' \bigcap J = \emptyset$ e $O' \bigcap O = \emptyset$ (ou seja, todo cluster é disjunto a qualquer outro cluster)

Assim, um cluster é definido por um grupo de cidades próximas e com o "monopólio" sobre certos objetos, de modo que sempre que algum destes objetos for recolhido, este deve ser recolhido neste cluster. Repare que por esta definição **se um objeto está disponível em alguma cidade, então ele estará disponível em apenas um único cluster**.

Idealmente γ é pequeno se comparado com o tamanho das arestas que ligam cidades do cluster com as cidades externas ao cluster, de modo que sempre que o ladrão visite alguma cidade do cluster, ele aproveite e e visite as demais cidades do cluster já que estas se encontram próximas umas das outras. Da mesma forma, se ele desejar retirar algum objeto o_l deste cluster, ele deve retirá-lo na última cidade do cluster visitada, tal que este objeto esteja disponível nesta cidade. Abaixo será dado um exemplo de aplicação deste conceito para clusters com valores pequenos de γ.

Exemplos

Inicialmente, considere o TTP1 dado na Figura 16 que consiste em 9 cidades distribuídas em $N_c = 3$ conjuntos de 3 cidades cada. Ao lado de cada ponto da figura está o número da cidade correspondente a este ponto, bem como os objetos disponíveis em cada cidade (entre {}). Estes aglomerados de cidades serão chamados de *clusters* 1, 2 e 3. Note que o objeto {4} somente se encontra disponível no cluster 1, enquanto o objeto {1} está disponível no cluster 3, e os objetos {2} e {3} se encontram no cluster 2. Os demais parâmetros deste TTP1 como o peso máximo da mochila, velocidades máxima e mínima, e etc não serão importantes para este exemplo e serão ignorados. Note que estes clusters indicados se adequam à definição dada anteriormente.

Defina a **distância entre dois clusters** como a menor aresta que conecta duas cidades destes clusters (uma de cada cluster). Tendo essa definição em mãos, destaque as cidades de cada cluster que estão mais próximas aos outros clusters[1]. Isso é exemplificado na Figura 17 onde por exemplo, a cidade {9} é a cidade do cluster 3 mais próxima ao cluster 2, assim como a cidade {6} é a cidade do cluster 2 mais próxima ao cluster 3. Na figura abaixo estas cidades estão destacadas em vermelho.

Estas cidades são definidas como **cidades representantes** de cada cluster, de

[1] Estas cidades devem ser diferentes. Caso uma mesma cidade seja a mais próxima a dois clusters diferentes, escolha um destes clusters para ser relacionado a esta cidade e em seguida busque a outra cidade mais próxima ao outro cluster.

Figura 16 – Um TTP1 organizado em clusters.

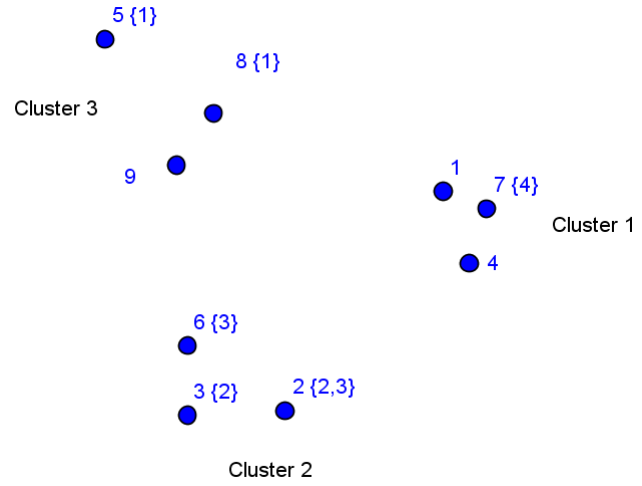
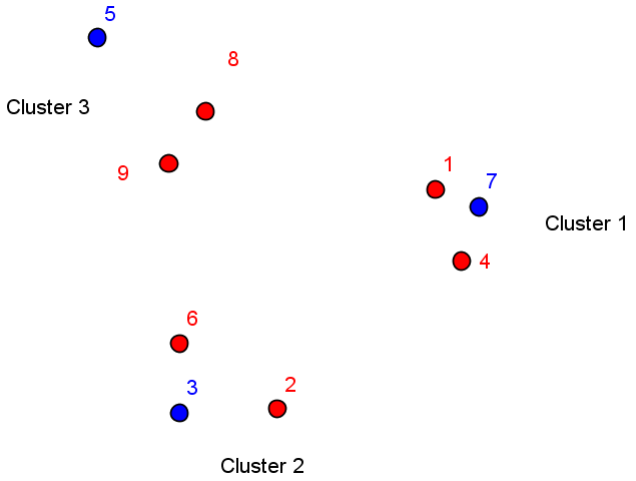


Figura 17 – Cidades representantes de cada cluster.



modo que a matriz *Rep* é definida como uma matriz $N_c \times N_c$ tal que se $c \in C$ é a cidade representante do cluster i com relação ao cluster j[2] então $Rep(i, j) = c$. Neste exemplo tem-se que:

$$Rep = \begin{pmatrix} - & 4 & 1 \\ 2 & - & 6 \\ 8 & 9 & - \end{pmatrix} \tag{5.1}$$

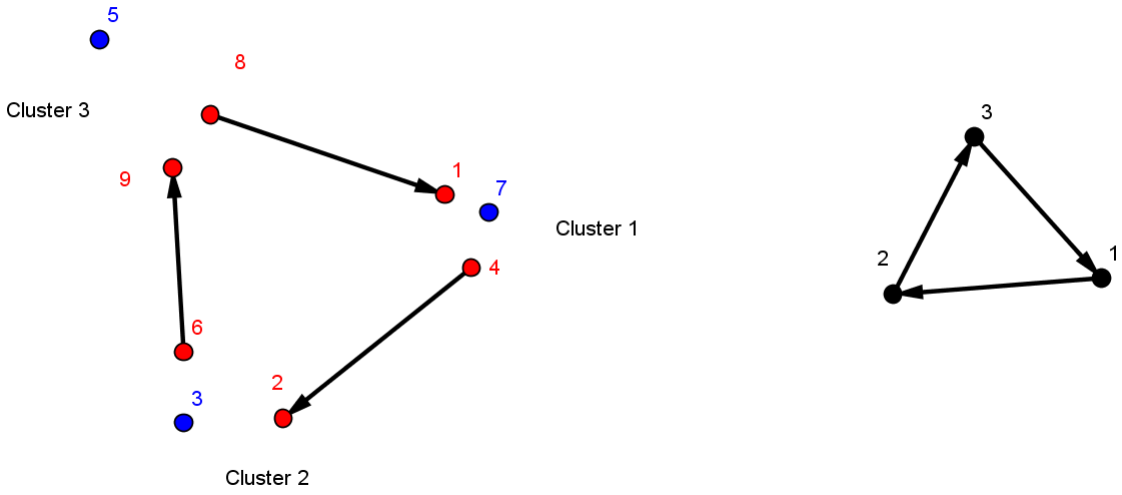
Estas cidades representantes serão importantes pois sempre que o ladrão se loco-

[2] Ou seja, c é a cidade do cluster i mais próxima ao cluster j (levando em conta o que foi dito na nota de rodapé anterior a esta).

mover entre clusters ele saiba qual a cidade mais próxima. Assim, caso o ladrão queira visitar as cidades do cluster 2 e logo após, as cidades do cluster 3, ele sabe que a última cidade do cluster 2 que ele deve visitar será a cidade $\{6\}$ (a mais próxima ao próximo cluster a ser visitado) e logo em seguida ele deve visitar a cidade $\{9\}$ (a mais próxima ao cluster em que ele se encontra) e assim ele chegará ao cluster 3. O caminho $\bar{x}_c = (1, 2, 3, 1)$ entre clusters pode ser representado pela Figura 18.

Assim, partindo do cluster 1, em direção ao cluster 2, o ladrão deve tomar a aresta correspondente: $d_{Rep(1,2),Rep(2,1)}$. Da mesma forma, ao sair do cluster 2 em direção ao cluster 3, o ladrão deve tomar a aresta $d_{Rep(2,3),Rep(3,2)}$, e finalmente, ao voltar ao cluster 1 a partir do cluster 3 o ladrão deve tomar a aresta $d_{Rep(3,1),Rep(1,3)}$.

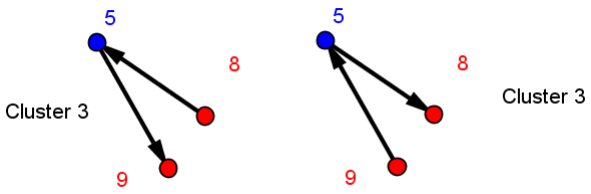
Figura 18 – Cidades representantes no caminho $\bar{x}_c = (1, 2, 3, 1)$ e simplificação deste caminho utilizando os clusters como vértices.



Assim, um caminho entre clusters como $\bar{x}_c$ é quase equivalente a algum caminho entre todas as cidades, faltando completar a Figura 18 com as arestas que ligam as demais cidades. Para tal, calculam-se os caminhos mínimos que ligam as cidades representantes de cada cluster (duas a duas) e percorrem todas as cidades do cluster. A Figura 19 mostra todos os caminhos relacionados ao cluster 3.

Assim, para um TTP1 simplificado em N_c clusters ($N_c > 2$) têm-se um total de $N_c - 1$ cidades representantes para cada cluster, e portanto $\frac{(N_c - 1) \times (N_c - 2)}{2}$ caminhos a serem calculados em cada cluster, e portanto será um total de $\frac{N_c \times (N_c - 1) \times (N_c - 2)}{2}$

Figura 19 – Caminhos que ligam as cidades representantes do cluster 3.



caminhos calculados. O tempo necessário para calcular cada caminho depende do número de cidades existentes em cada cluster, de modo que um cluster com $n_c \leq n$ cidades possui $(n_c - 2)!$ possíveis caminhos interligando cada par de cidades representantes. Caso este número de caminhos seja muito grande para ser calculado por força bruta, há muitas formas de calculá-los que já foram estudadas na literatura, pois isto se trata de uma variante do TSP[3] [6].

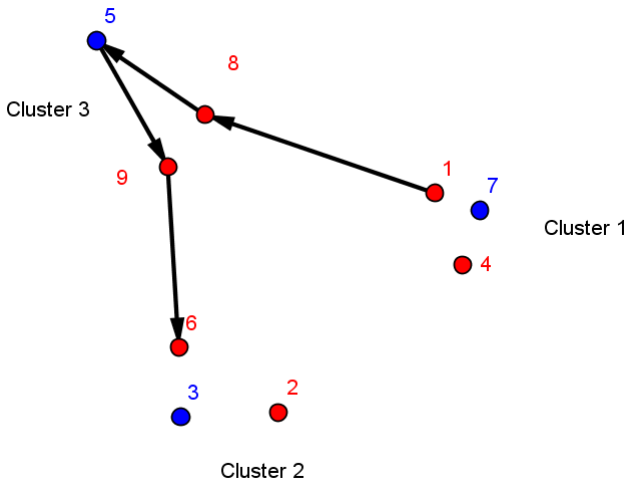
Tendo em mãos os caminhos relacionados ao cluster 3 por exemplo (que podem ser visualizados na Figura 19) o ladrão sabe por exemplo que, ao sair do cluster 1, chegar ao cluster 3 e percorrer suas cidades, e em seguida seguir viagem até o cluster 2, então ele deve percorrer o caminho mostrado na Figura 20 abaixo, que basicamente completa o caminho (1,3,2) entre clusters com um caminho interno ao cluster 3, percorrendo todas as suas cidades.

Portando, o caminho entre clusters[4] $\bar{x'}_c = (1, 3, 2, 1)_c$ pode ser traduzido pelo caminho entre cidades $\bar{x'} = (1, 8, 5, 9, 6, 3, 2, 4, 7, 1)$, como é ilustrado na Figura 21. Esta conversão de caminhos entre clusters para caminhos entre cidades será importante para a simplificação de TTP1's onde, ao invés de se trabalhar com um número grande de cidades, pode-se trabalhar com um número relativamente menor de clusters. No caso do exemplo do Capítulo 4 mostrado na Figura 10 seriam necessários 5 clusters, sendo 3 deles compostos por 25 cidades, um composto por 24 cidades, e outro composto por apenas uma cidade (cidade $\{1\}$), de modo que o primeiro cromossomo de uma solução deste exemplo seria

[3] Aqui se trata do GTSP (*Generalized Travelling Salesman Problem*) que separa as cidades em clusters e buscas caminhos Hamiltonianos em cada cluster, que no caso da simplificação do TTP1, busca um caminho com vértices inicial e final prefixados.

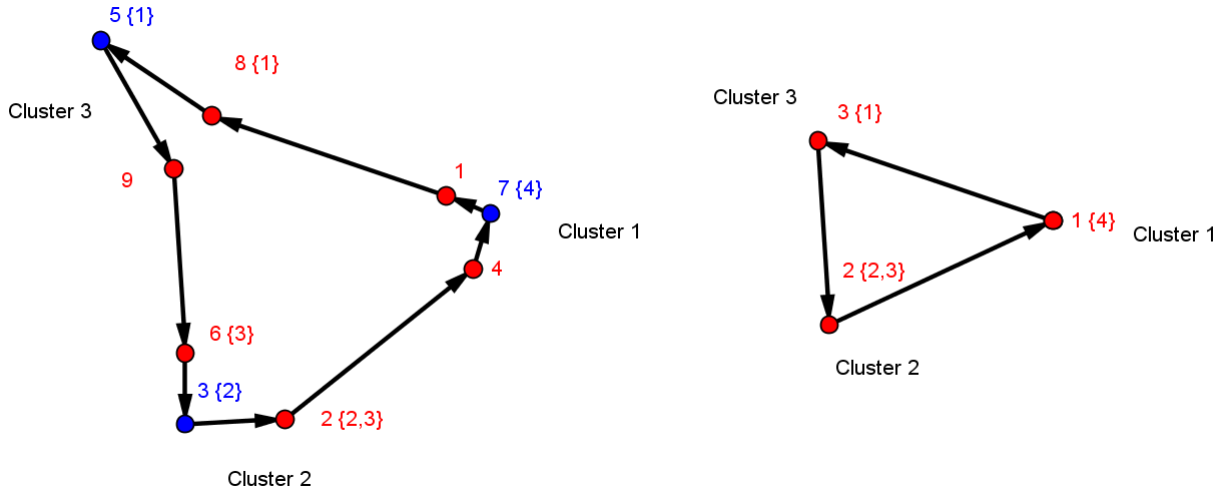
[4] A partir daqui será utilizado o subíndice 'c' para enfatizar que os algarismos utilizados representam clusters e não cidades. Assim os algarismos {1,2,3,4} se referem aos clusters correspondentes, e não às cidades 1,2,3 ou 4.

Figura 20 – Aplicação dos caminhos entre cidades representantes.



representada por $N_c + 1 = 6$ genes, ao invés dos $n + 1 = 101$ genes usuais[5].

Figura 21 – Caminho na forma simplificada $\bar{x'}_c = (1, 3, 2, 1)_c$ e na forma expandida $\bar{x'} = (1, 8, 5, 9, 6, 3, 2, 4, 7, 1)$.



Falta agora decidir o que fazer quanto aos objetos de cada cluster de modo a representar uma solução genérica em clusters $(\bar{x}_c, \bar{z}_c)$ como uma solução no formato tradicional $(\bar{x}, \bar{z})$.

Suponha a solução $(\bar{x'}_c, \bar{z'}_c) = ((1, 3, 2, 1)_c, (3, 2, 2, 1)_c)$ para o problema simplificado em clusters. Sabe-se que o caminho equivalente a $\bar{x'}_c$ percorrendo as cidades do TTP1

[5] Vale lembrar que nem todo caminho possível pode ser representado utilizando um caminho em clusters, mas isto é algo desejável se as cidades deste cluster se encontram distantes dos demais clusters, como será visto com mais detalhes no Capítulo 6.

original é $\bar{x'} = (1, 8, 5, 9, 6, 3, 2, 4, 7, 1)$, faltando descobrir qual o $\bar{z'}$ correspondente a $\bar{z'}_c$. Observe em $\bar{z'}_c$ que o objeto 1 foi retirado do cluster 3, e portanto este deve ser retirado da cidade $\{5\}$ ou da cidade $\{8\}$. Como os objetos a serem retirados já foram escolhidos, então o único parâmetro que pode ser controlado a fim de se obter uma função-objetivo máxima é a ordem de retirada destes objetos. Quando um objeto é retirado de alguma cidade ele reduz a velocidade do ladrão durante o resto do percurso, logo fixando a retirada de um objeto o melhor a se fazer é postergar esta retirada ao máximo, de modo a reduzir o tempo gasto no percurso escolhido. Assim, a melhor cidade a ser escolhida para se retirar este objeto é a cidade $\{5\}$.

Usando os mesmos argumentos conclui-se que as melhores cidades para se retirar os demais objetos são as cidades $\{2\}$ (para a retirada dos itens 2 e 3 do cluster 2) e $\{7\}$ (para a retirada do item 4 do cluster 1). Assim define-se que a simplificação $\bar{z'}_c = (3, 2, 2, 1)_c$ corresponde a $\bar{z'} = (5, 2, 2, 7)$.

Finalmente, defina a transformação[6]:

$$\begin{aligned} T_c : \mathbb{N}^{N_c+1} \times \mathbb{N}^m &\to \mathbb{N}^{n+1} \times \mathbb{N}^m \\ (\bar{x}_c, \bar{z}_c) &\to T_c(\bar{x}_c, \bar{z}_c) = (\bar{x}, \bar{z}) \end{aligned} \tag{5.2}$$

Onde $(\bar{x}_c, \bar{z}_c)$ é uma solução representada na forma de um caminho entre clusters e um plano de coleta também denotado por índices de clusters, e $(\bar{x}, \bar{z})$ é a solução equivalente no domínio do problema original composto por um caminho entre cidades do problema, e um plano de coleta de objetos também relacionado a cidades. Esta transformação deve ser feita (como exemplificado acima) na seguinte ordem:

- Denote o caminho entre clusters como um caminho entre cidades representantes, como feito na Figura 18.
- Complete o caminho $\bar{x}$ com os caminhos mínimos que ligam as cidades representantes, como calculado na Figura 19 e implementado na Figura 20.
- Escreva $\bar{z}$ de modo a escolher as cidades mais próximas ao final do caminho $\bar{x}$ que estejam contidas nos clusters escolhidos por $\bar{z}_c$, como exemplificado acima.

[6] Sem perda de generalidade, serão usados apenas números naturais para denotar cidades, objetos e clusters.

Tendo em mãos este novo operador, é possível simplificar um problema com 100 cidades como o problema da Figura 10 em um problema com apenas 5 clusters como na Figura 15, de modo que agora há apenas $(5-1)! = 24$ possíveis valores para $\bar{x}_c$, enquanto há apenas[7] $2^m = 2^7 = 128$ possíveis valores para $\bar{z}$, logo há $24 \times 128 = 3072$ possíveis pares $(\bar{x}_c, \bar{z}_c)$, de modo que o problema simplificado pode ser facilmente resolvido por força bruta[8]. Se a maior função-objetivo encontrada por esta simplificação é maior que aquela encontrada pelo algoritmo genético construído no Capítulo 4 dependerá do TTP1 estudado.

Aplicação

O objetivo principal da definição dos clusters é o de aproveitar características inerentes a certos problemas que não seriam utilizadas pelo algoritmo genético.

Como explicado na seção 2.4 um algoritmo genético é um algoritmo de uso geral completamente cego às características do problema trabalhado. Um algoritmo genético não vê diferenças (a nível de execução) entre o problema ilustrado pela Figura 10 com um TTP1 completamente arbitrário. Com a transformação T_c entre caminhos entre clusters e caminhos entre cidades ilustrada na Figura 21 é possível transformar o problema trabalhado em outro problema, de modo que este novo problema pode ser resolvido por um algoritmo genético (ou mesmo por força bruta, caso o número de clusters e objetos seja pequeno) que, mesmo sendo "cego" ao fato do problema inicial ser organizado em clusters, ainda irá explorar esta característica do problema inicial devido à transformação T_c.

Um novo algoritmo genético foi criado para a utilização desta transformação, de modo que seja possível comparar o algoritmo genético original com este novo algoritmo e avaliar até que ponto esta simplificacao de um problema em clusters é interessante. Este novo algoritmo consiste em uma cópia do algoritmo 3 mas com uma das entradas diferente: ao invés de utilizar um TTP1 como entrada, ele utiliza um TTP1_c que nada mais é do que uma transformação do TTP1 original da seguinte forma:

[7] Por definição um objeto está disponível em um, e apenas um cluster, logo para cada entrada de $\bar{z}_c$ há apenas duas possibilidades: o objeto ser retirado do seu respectivo cluster (e portanto esta entrada terá valor igual ao índice deste cluster) ou este objeto não foi retirado (e portanto esta entrada terá valor nulo).

[8] Isto supondo que a matriz Rep já foi calculada, bem como os caminhos que ligam cada par de cidades representantes de cada cluster (o que tem um alto custo computacional).

- As "cidades" do TTP1_c são clusters, e as distâncias entre estas "cidades" é dada pela distância entre estes clusters que é a distância entre suas cidades representantes.

- Os itens estão distribuídos entre os clusters deste problema, que representam suas "cidades".

- A função-objetivo é calculada de forma diferente, de modo que a aptidão de uma solução $(\bar{x}_c, \bar{z}_c)$ é dada pela aptidão de $T_c(\bar{x}_c, \bar{z}_c)$, de modo que sempre que for necessário calcular a aptidão de um indivíduo ele deve ser convertido por T_c primeiro.

O algoritmo novo será portanto desta forma:

Algoritmo 4: Algoritmo genético utilizado

Entrada: TTP1_c, parâmetros do algoritmo genético
1 Populacao de soluções $t \leftarrow 0$
2 **Gere população inicial** Pc_t **com** $\mathbf{N_g}$ **indivíduos**
3 **Converta** Pc_t em seu equivalente $T_c(Pc_t) = P_t$
4 **Avalie** Pc_t e use estes valores como a aptidão de Pc_t
5 Critérios de parada não forem satisfeitos **Selecione** $\mathbf{N_g}$ **pares** de indivíduos progenitores
6 **Cruze** os progenitores (com **mutação**), obtendo uma nova população Fc_t de $\mathbf{N_g}$ filhos
7 **Converta** Fc_t em seu equivalente $T_c(Fc_t) = F_t$
8 **Avalie** F_t e use estes valores como a aptidão de Fc_t
9 **Selecione** os indivíduos de $Pc_t \bigcup Fc_t$, que farão a nova geração Pc_{t+1}
10 **Converta** Pc_{t+1} em seu equivalente $T_c(Pc_{t+1}) = P_{t+1}$
11 **Avalie** P_{t+1} e use estes valores como a aptidão de Pc_{t+1}
12 $t \leftarrow t + 1$

O cálculo feito para descobrir quais são as cidades representantes de cada cluster, bem como o cálculo do caminho mínimo que liga cada par de cidades representantes foram feitos previamente à execução do algoritmo em cada simulação apresentada neste estudo, de modo que estes caminhos são armazenados na memória do computador e sempre que a conversão de um indivíduo for feita, é necessária uma consulta na memória para que a função de conversão saiba como converter cada indivíduo.

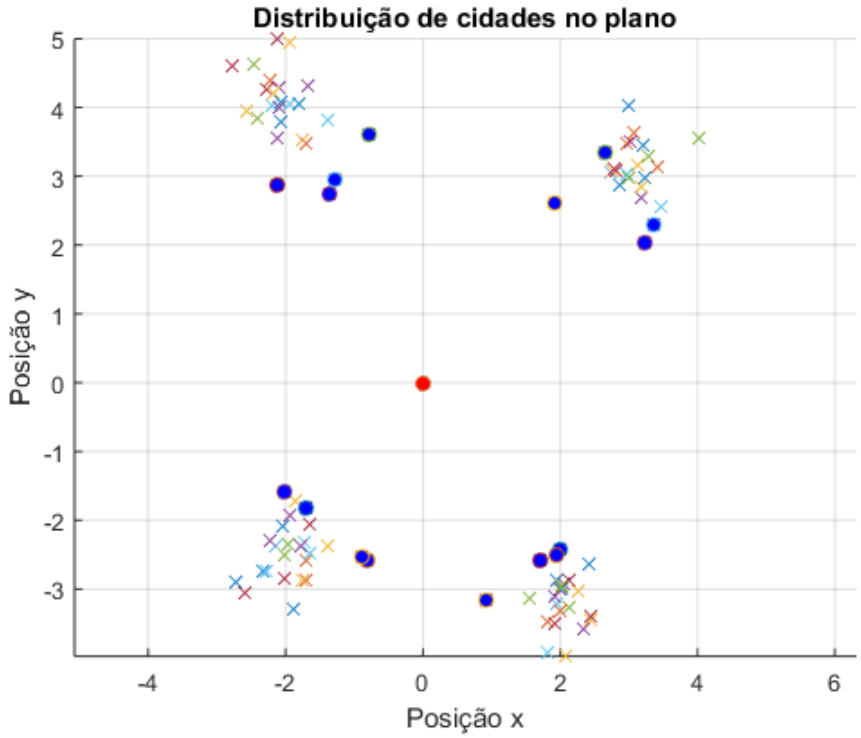
A título de comparação foram feitas 10 simulações do problema da Figura 10. A divisão dos clusters é mostrada na Figura 15 de modo que a cidade $\{1\}$ compõe o cluster

1. A matriz de cidades representantes deste problema é mostrada a seguir:

$$Rep = \begin{pmatrix} - & 1 & 1 & 1 & 1 \\ 11 & - & 17 & 19 & 8 \\ 30 & 31 & - & 28 & 29 \\ 73 & 58 & 52 & - & 55 \\ 85 & 83 & 94 & 90 & - \end{pmatrix} \tag{5.3}$$

As cidades representantes são destacadas na figura abaixo:

Figura 22 – Cidades representantes do TTP1 exemplificado.



Note que a única representante do cluster 1 é a cidade $\{1\}$. Isto se deve ao fato de o cluster 1 ter sido escolhido como um cluster que possui apenas a cidade $\{1\}$, para que o formato $\bar{x}_c = (1, *, *, ..., *, 1)$ seja preservado, e o algoritmo genético construído anteriormente possa ser reutilizado (é claro, com modificações no cálculo da função-objetivo de cada indíviduo). Abaixo é ilustrada a evolução das aptidões dos indivíduos de cada simulação:

Na Figura 23 é mostrada a evolução da população de apenas a primeira das simulações:

Figura 23 – Aptidão dos indivíduos de cada simulação do algoritmo genético.

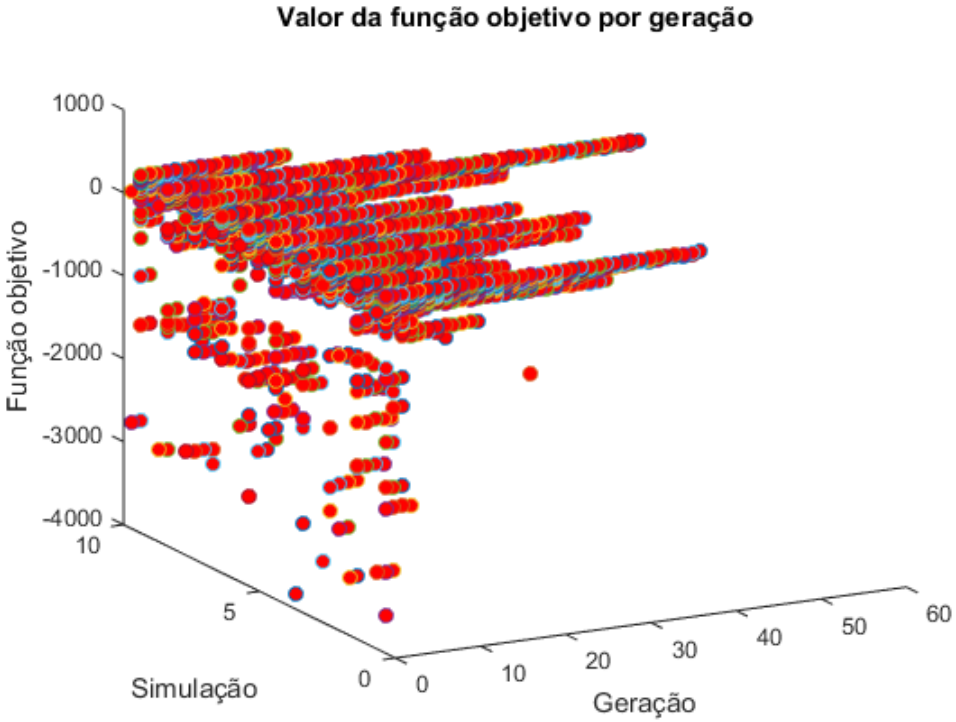
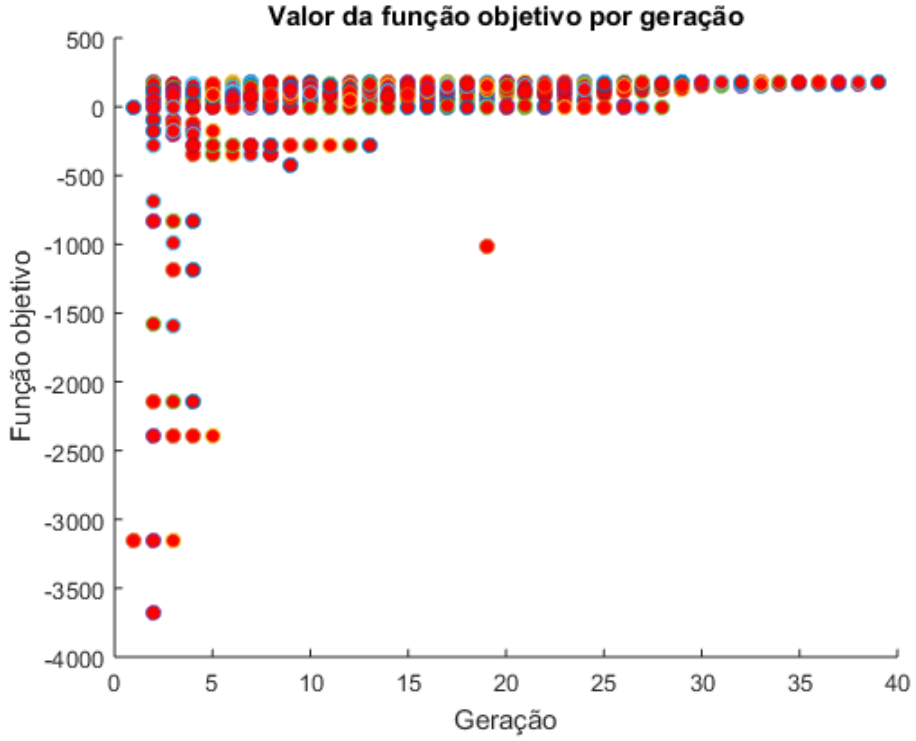
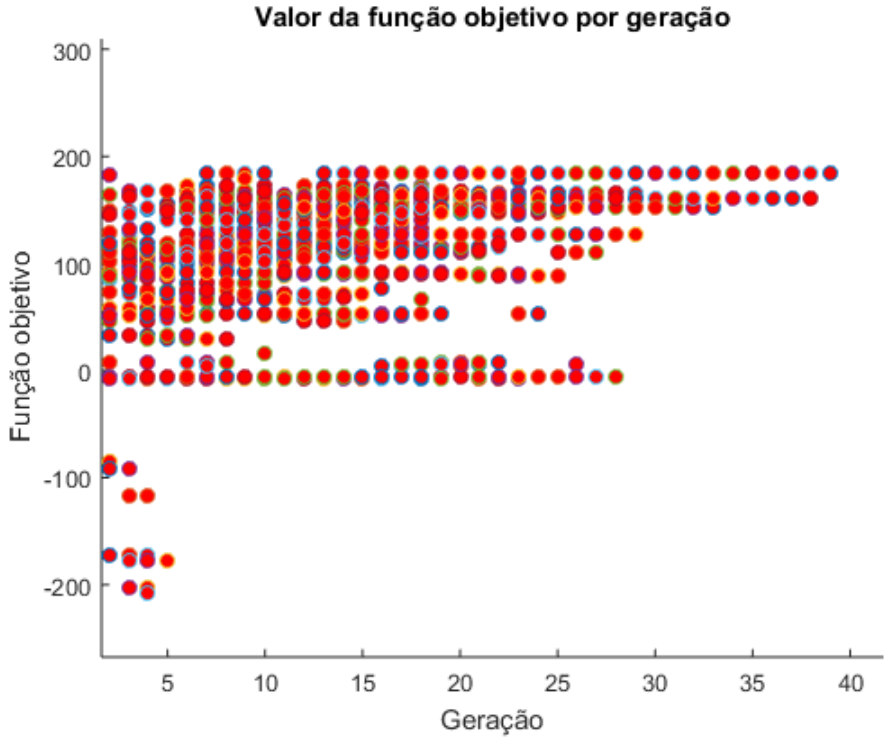


Figura 24 – Aptidão dos indivíduos da primeira simulação do algoritmo genético.



Aqui é possível observar esta evolução nas gerações finais com mais detalhes (foi feito um *zoom* na região superior da Figura 24):

Figura 25 – Convergência da aptidão dos indivíduos da primeira simulação.

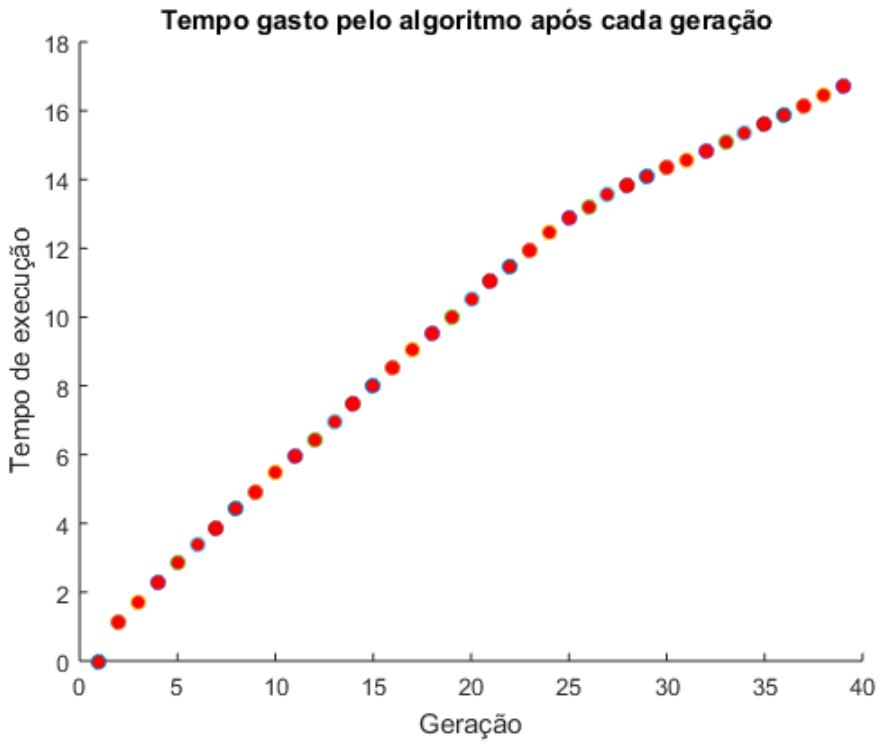


Abaixo é mostrado o tempo gasto pelo algoritmo. Foi utilizada a mesma máquina que nas simulações do algoritmo genético do capítulo anterior.

Note que, apesar do maior quantidade de gerações necessária para a convergência do algoritmo (39 gerações) o tempo total gasto na evolução destes indivíduos foi menor que o gasto no algoritmo genético original (foram gastos 16.70 segundos). Porém, foram necessários 414.63 segundos para o cálculo prévio de todos os caminhos mínimos que ligam cada par de cidades representantes de cada cluster.

Uma vantagem deste método é que após o cálculo dos caminhos mínimos que ligam cada par de cidades representantes em cada cluster, o tempo gasto para executar uma única simulação é drásticamente reduzido, de modo que foram gastos 118.43 segundos para executar as 10 simulações mostradas, totalizando $118.43 + 414.63 = 533.06$ segundos de simulação, enquanto as 10 simulações feitas no capítulo anterior precisaram de 1889 segundos para serem feitas. Repare que assim como na Figura 14 do Capítulo 4 o tempo gasto pelo algoritmo para gerar cada geração a partir da 25ª geração foi reduzido, indicando

Figura 26 – Tempo gasto para executar o algoritmo.



que o algoritmo encontrou um mínimo local. Neste algoritmo são necessárias mais gerações para se chegar a este mínimo, porém o tempo gasto para se gerar cada nova geração é bem menor (aproximadamente 0.2s por geração, enquanto A1 precisa de pelo menos 15s por geração).

Abaixo é mostrada uma das soluções encontradas na geração final desta simulação:

Tabela 14 – Uma solução para o problema simplificado.

$$\bar{x}_c = \quad (1, \quad 3, \quad 5, \quad 4, \quad 2, \quad 1)$$

$$\bar{z}_c = \quad (2, \quad 2, \quad 4, \quad 0, \quad 3, \quad 0, \quad 0, \quad 0)$$

Seu valor convertido é mostrado em 5 partes pelas tabelas abaixo. A Tabela 20 mostra o seu correspondente $\bar{z}$.

1	30	28	38	33	27	48	40	44	31	43	26	42	47	34	36	45	41	37	32

Tabela 15 – Primeiras 20 entradas de $\bar{x} = T_c(\bar{x}_c)$ (ou melhor, $\bar{x}_c$ convertido).

Repare por exemplo que o caminho $\bar{x}_c = (1, 3, 5, 4, 2, 1)$ parte do cluster 1 em direção ao cluster 3, observando a matriz Rep conclui-se que o ladrão deve tomar a aresta

50	39	35	46	49	29	94	80	78	100	92	86	79	97	77	96	76	89	93	84

Tabela 16 – Entradas 21 a 40 de $\bar{x}$.

91	87	88	98	81	99	95	82	85	83	90	58	56	62	69	57	74	68	51	60

Tabela 17 – Entradas 41 a 60 de $\bar{x}$.

66	63	75	70	54	59	64	65	72	61	52	73	71	67	53	55	11	15	5	6

Tabela 18 – Entradas 61 a 80 de $\bar{x}$.

3	4	2	21	17	8	23	25	10	13	9	14	20	18	16	7	22	24	12	19

Tabela 19 – Entradas 81 a 100 de $\bar{x}$.

19	18	72	0	47	0	0	0

Tabela 20 – Valor de $\bar{z}$ correspondente.

$\overrightarrow{(Rep(1,3), Rep(3,1))} = \overrightarrow{(1,30)}$ que é a aresta que liga a cidade $\{1\}$ à cidade $\{30\}$. Após percorrer todo o cluster 3, o ladrão segue em direção ao cluster 5, e portanto deve tomar a aresta $\overrightarrow{(Rep(3,5), Rep(5,3))} = \overrightarrow{(29,94)}$. Isto é confirmado ao observar-se as tabelas (15) e (16) observando-se as primeiras 26 entradas de $\bar{x}$, e todo o caminho percorrido da cidade 30 (segunda entrada) à cidade 29 (25^{a} entrada) é exatamente o caminho mínimo que liga estas duas cidades representantes e que percorre todas as cidades do cluster 3 (cidades 26 a 50). O mesmo pode ser observado nos demais trechos de $\bar{x}$.

Neste exemplo foi observado um menor tempo gasto para se executar as 10 simulações do algoritmo via simplificação por clusters, e além disso as aptidões dos indivíduos encontrados foi maior do que as aptidões encontradas pelo algoritmo genético genérico. Obviamente este exemplo foi escolhido para demonstrar quando esta abordagem se mostra interessante, mas também é necessário avaliar até que ponto esta abordagem se mostra atraente. No capítulo seguinte serão estudados outros tipos de problemas de modo a avaliar de forma empírica até que ponto a simplificação se mostra eficaz.

6 Resultados e Discussão

Neste capítulo serão feitas simulações de diferentes problemas com clusters no mesmo formato dos exemplos vistos no capítulo anterior, de modo a comparar o desempenho do algoritmo genético de uso geral apresentado no Capítulo 4 com o algoritmo genético aplicado ao problema simplificado na forma mostrada no Capítulo 5.

Na seção 6.1 são apresentados os problemas que serão utilizados para comparar os dois algoritmos. São mostrados seus principais parâmetros bem como os principais critérios utilizados para a escolha destes problemas.

Na seção 6.2 são mostrados os resultados das simulações feitas, comparando os dois algoritmos quando utilizados para resolver cada um dos problemas apresentados. Para cada problema são feitas 10 simulações de cada algoritmo e em seguida são comparados os valores da função-objetivo e o tempo de execução dos algoritmos nestas simulações.

Na seção 6.3 é feita uma discussão sobre os resultados obtidos na seção 6.2. São comparados os resultados obtidos em cada algoritmo de modo a discutir quando um tipo de algoritmo pode ter um desempenho melhor do que o outro.

No capítulo seguinte será feita uma conclusão deste estudo, apresentando um resumo do que foi feito, as principais conclusões obtidas e quais são os próximos passos a serem tomados a fim de continuar este estudo com mais profundidade.

6.1 Os problemas

Com o objetivo de se comparar o algoritmo genético do Capítulo 4 com sua aplicação utilizando a simplificação de um problema em clusters, como foi feito no Capítulo 5, foi escolhida uma variação do problema tratado nos capítulos anteriores. Esta variação será chamado de problema **P1** e consiste em um TTP1 composto por 200 cidades e 8 objetos distribuídos entre 9 clusters, de modo que o cluster 1 (composto somente pela cidade {1}) não possui objetos disponíveis, enquanto cada um dos outros 8 clusters pussui um objeto cada. Os clusters 2 a 9 estão distribuídos no plano conforme ilustrado na Figura 27 que também mostra quais são as cidades representantes de cada um destes clusters (marcados

por círculos azuis) indicando cada cluster por seu respectivo índice.

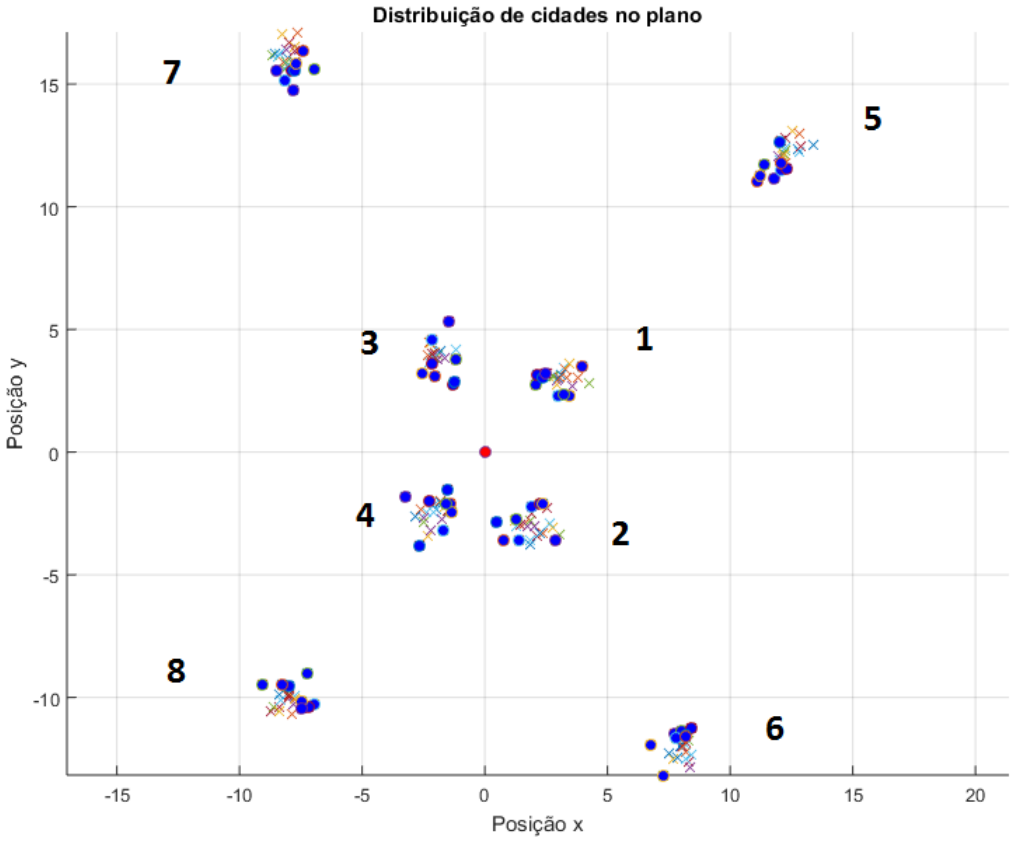


Figura 27 – Distribuição das cidades no problema P1.

Cada cluster é composto por 25 cidades com exceção do cluster 1, que é composto apenas pela cidade {1}, e o cluster 2 que é composto por 24 cidades. Assim como no exemplo trabalhado, todos os clusters são compostos por cidades aleatoriamente[1] distribuídas em círculos de raio $\frac{\pi}{2}$. O algoritmo utilizado para gerar este problema também decide se será ou não colocado algum objeto em alguma cidade, quando esta é criada[2] (i.e, quando sua posição no plano é gerada).

Os clusters 1 a 5 estão centrados nas mesmas posições do exemplo estudado nos

[1] Esta distribuição foi feita da seguinte forma: Para cada cidade em um cluster, escolhem-se aleatoriamente, de forma uniforme, dois números reais **a** e **b** entre 0 e 1 utilizando a função **rand()** do MATLAB. Assim, se este cluster for centrado em (0,0) seria inserida uma cidade com posição $(a \times \frac{\pi}{2}, b \times 2\pi)$ em coordenadas polares. Se este cluster estiver centrado em outro ponto do plano, é feita a devida adaptação.

[2] Para cada cidade é dada uma chance de 10% de esta ser escolhida como uma das cidades que possuirá algum objeto. As cidades do i-ésimo cluster só podem possuir o (i-1)-ésimo item, de modo que o cluster 1 não possui itens disponíveis para retirada.

capítulos anteriores: (0,0), (3,3), (2,-3), (-2,4) e (-2, -2.5) respectivamente. O cluster 6 está centrado em $(12, 12) = 4 \times (3, 3)$ de modo que seu centro é colinear com o centro dos clusters 1 e 2, estando 4 vezes mais distante da origem do plano do que o cluster 2. Analogamente, os clusters 7 a 9 estão centrados em (8,-12), (-8,16) e (-8, -10) respectivamente. O principal objetivo por trás desta escolha é o de avaliar o desempenho dos dois algoritmos em problemas onde os clusters estão distribuídos em regiões distantes, e em seguida, estes clusters serão aproximados (vide problemas P2, P3 e P4) de modo que a distinção entre alguns clusters pode ser difícil se for avaliada apenas a posição espacial de suas cidades. Em seguida, estes clusters serão ainda mais aproximados (vide problemas P5 e P6) de modo que todas as cidades se encontrem embaralhadas em torno da cidade {1}.

Assim, foram gerados também os problemas P2 a P6, que são problemas distindos entre si (e com relação ao problema P1) mas com a característica de que seus clusters 1 a 5 possuem cidades distribuídas em posições diferentes de problema para problema, mas cada cluster possui o mesmo número de cidades em cada problema, e são centradas nas mesmas posições (e distribuídas em círculos de mesmo raio, $\frac{\pi}{2}$). Os clusters 6 a 9 se encontram com centros com fatores (com relação ao centro dos clusters 2 a 5) 3, 2, 1 nos problemas P2, P3 e P4 respectivamente. Nos problemas P5 e P6 foram utilizados fatores 0.5 e 0.1 em todos os clusters, de modo que no problema P5 os clusters 2 e 6 possuem centros localizados em (1.5,1.5), e analogamente os clusters 3 e 7 estão centrados em (1,-1.5), os clusters 4 e 8 em (-1,2) e os clusters 5 e 9 em (-1,-1.25). Da mesma forma, os mesmos pares 2 e 6, 3 e 7, 4 e 8, e, 5 e 9 se encontram centrados em (0.3,0.3), (0.2,-0.3), (-0.2,0.4), e (-0.2,-0.25) respectivamente.

As figuras (28), (29), (30), (31) e (32) ilustram a distribuição das cidades de cada problema destacando suas cidades representantes.

Devido ao grande tamanho das matrizes A (matriz de 8x200 entradas) e a matriz de distâncias (matriz de 200x200 entradas) seus valores serão omitidos. Da mesma forma, serão omitidos os caminhos mínimos que interligam cada par de cidades representantes, de modo que na seção 6.2 serão apresentados apenas os resultados obtidos pelas simulações.

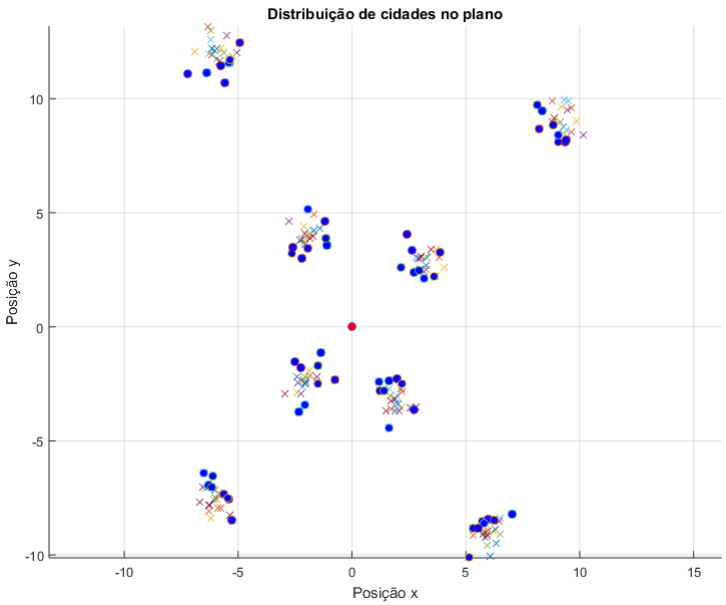


Figura 28 – Distribuição das cidades no problema P2.

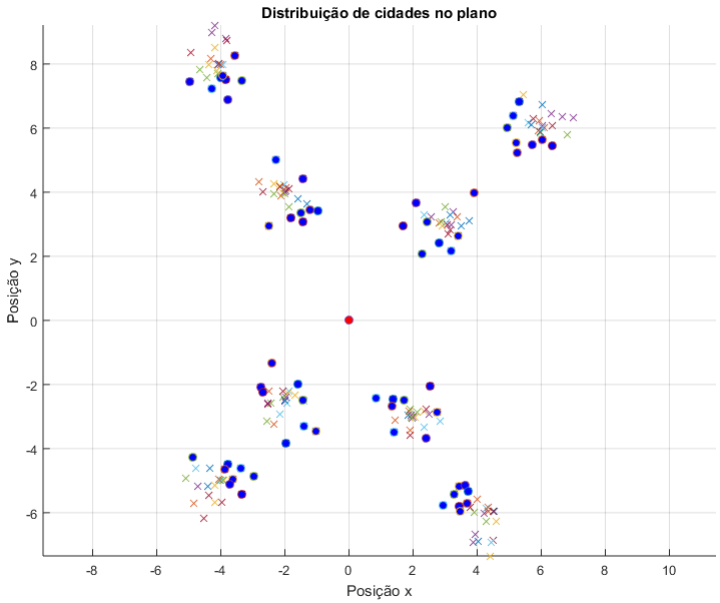


Figura 29 – Distribuição das cidades no problema P3.

6.2 Resultados

A seguir serão apresentados os principais resultados obtidos pelas simulações feitas. Para simplificar a notação, o algoritmo genético puro será chamado de Algoritmo

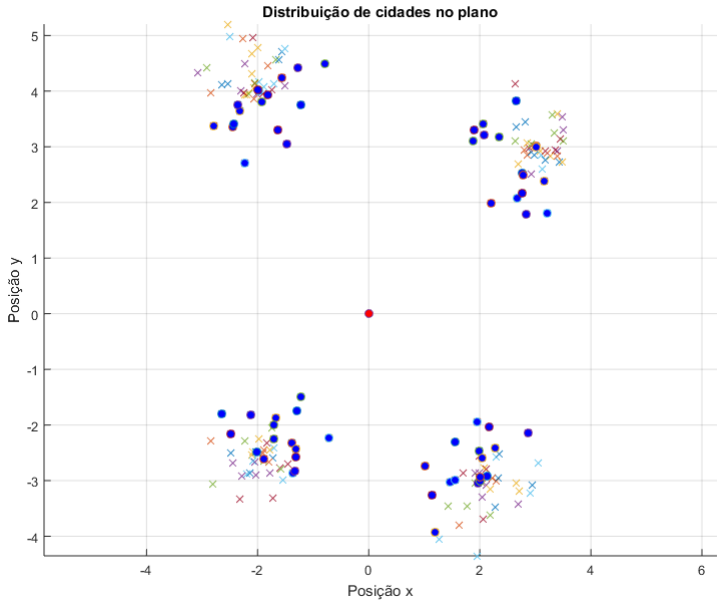


Figura 30 – Distribuição das cidades no problema P4.

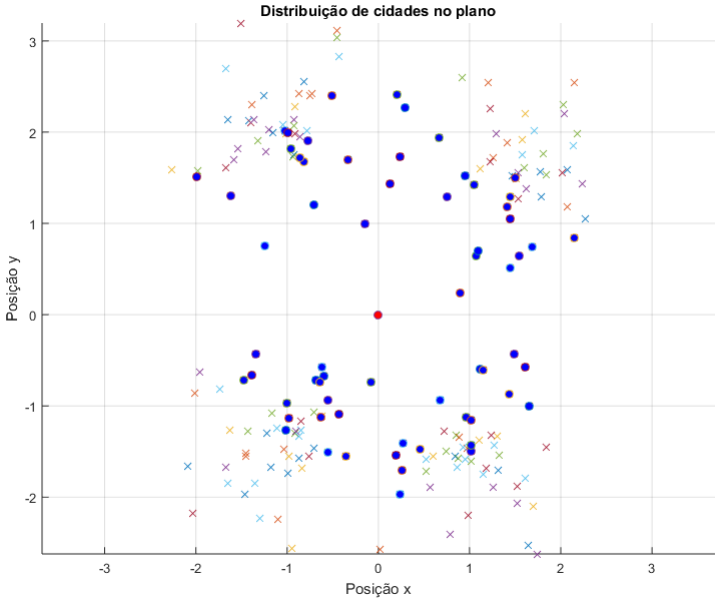


Figura 31 – Distribuição das cidades no problema P5.

A1, enquanto que o algoritmo genético aplicado à versão simplificada de um problema (utilizando a abordagem via clusters) será chamado de Algoritmo A2. Para cada problema enunciado na seção anterior foram feitas 10 execuções do Algoritmo A1, e 10 execuções

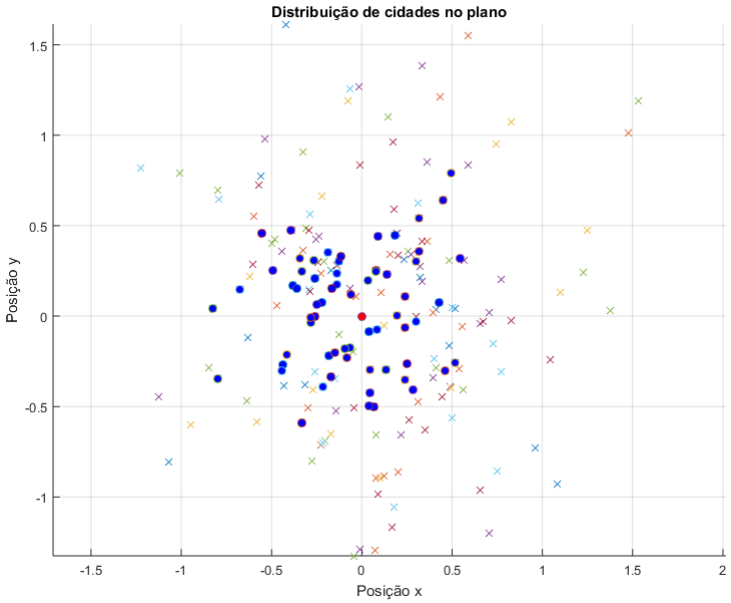


Figura 32 – Distribuição das cidades no problema P6.

do Algoritmo A2. Na figura[3] Figura 33 são mostrados os valores máximos das aptidões dos indivíduos dentre todas as gerações em cada uma das simulações feitas, e para cada problema. Os elementos em azul correspondem a A1 e os elementos em vermolho a A2. Círculos representam os valores máximos de cada simulação de A1 enquanto asteriscos representam os valores máximos obtidos por A2.

Na Figura 33 são mostrados os valores máximos das aptidões dos indivíduos dentre todas as gerações em cada uma das simulações feitas, e para cada problema.

Repare que os resultados obtidos pelo Algoritmo A2 se mostram melhores, mas a eficácia de A1 se aproxima de sua eficácia à medida que as medições vão do problema P1 ao problema P6. As linhas contínuas desenhadas mostram a evolução dos valores médios de todas as simulações em cada problema, enquanto as linhas pontilhadas mostram a evolução do maior valor obtido dentre todas as simulações de cada problema. Vale lembrar que cada problema tratado é diferente dos demais, e portanto a melhor solução (bem como a aptidão) de cada problema pode variar, mas vale destacar que A1 começa a ser mais eficaz à medida que os clusters deixam de ser isolados (como em P1) e começam a se misturar com os demais clusters (como em P6).

[3] Uma descrição mais detalhada dos elementos desra figura se encontram no texto. Isto foi feito devido ao grande número de elementos que estariam em sua legenda, e por isto foi decidido simplificá-la.

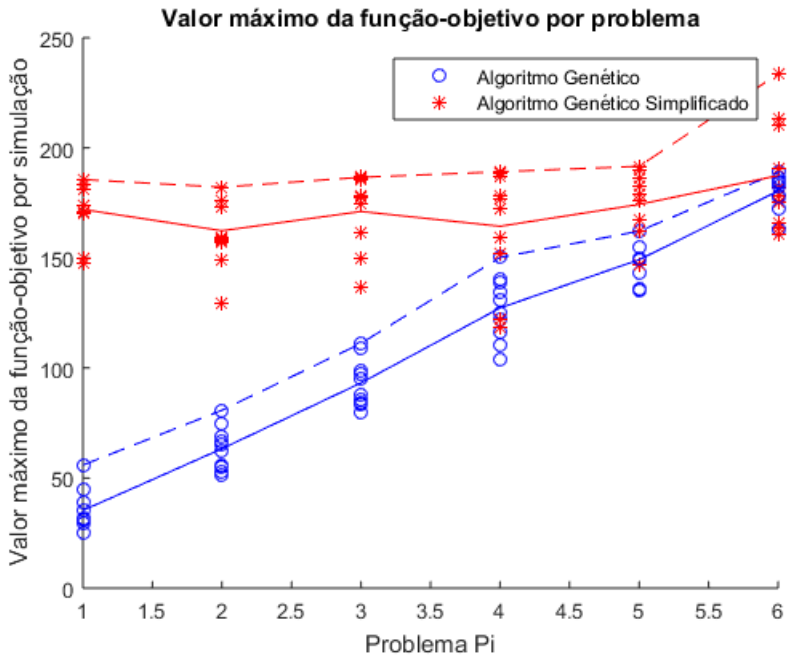


Figura 33 – Aptidão máxima dos indivíduos de cada simulação.

Na Figura 34 é mostrada a evolução das relações $\frac{Valor médio obtido em A1}{Valor médio obtido em A2}$ e $\frac{Valor máximo obtido em A1}{Valor máximo obtido em A2}$.

Para a obtenção desta figura foi feita a média entre os resultados obtidos em cada uma das 10 simulações feitas em cada problema (e para cada algoritmo), e em seguida foi feita a divisão entre a média encontrada por A1 e a média encontrada em A2, obtendo o gráfico em azul. Já a curva vermelha foi obtida de forma semelhante, dividindo o máximo das simulações do algoritmo A1 pelo máximo obtido por A2. Como já podia ser visto na Figura 33 A2 encontrava melhores valores para a função-objetivo no Problema P1, mas os valores encontrados por A1 começaram a crescer à medida que foram mudados os problemas, de modo que no Problema P6 (onde a distinção entre clusters não é tão óbvia quanto em P1) os dois algoritmos são quase que indistinguíveis se for considerada apenas a aptidão média dos indivíduos encontrados, mas a aptidão máxima obtida por A2 continua superior à de A1.

Na Figura 35 é mostrado o tempo total gasto por cada algoritmo para a execução das 10 simulações de cada problema. Para A2 foram dados dois tempos: um variável, que

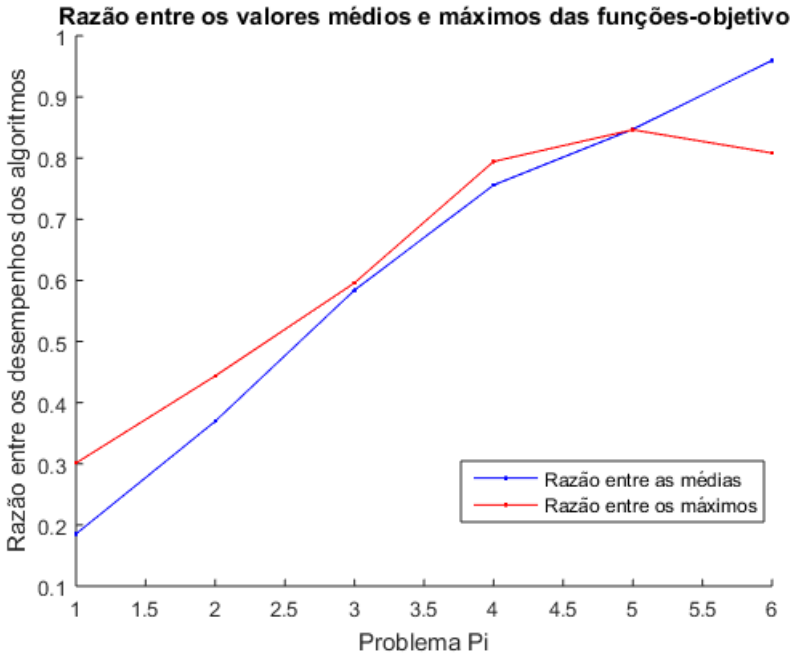


Figura 34 – Eficácia relativa dos algoritmos.

é o tempo gasto para o cálculo dos caminhos mínimos que ligam cada par de cidades representantes, e outro que é fixo e representa o tempo gasto pelo algoritmo para a execução das 10 simulações.

Note que o tempo gasto por A2 sempre é menor que o tempo gasto por A1 para a execução de 10 simulações, mas o tempo fixo de A2, que passa a marca dos 2000 segundos, é maior do que o tempo necessário para que A1 execute apenas uma iteração (que em média, para P1 por exemplo, demoram menos do que 1400 segundos).

Também foram feitas simulações de variações de P6 com o fim de comparar o desempenho de A1 e A2 para um problemas de estrutura semelhante (mesmo número de clusters com a mesma distância entre si) mas variando o número de cidades do problema. Assim, foram obtidos os problemas P6.2, P6.3, P6.4, P6.5 e P6.6, que (assim como P6) possuem 9 clusters: 7 clusters com K cidades, um com K-1 cidades, e um com 1 cidade. Enquanto P6 possui K = 25, P6.2 a P6.6 possuem, respectivamente[4], K = 25, 30, 35, 40, 45. Estes problemas possuem 200, 240, 280, 320, e 360 cidades respectivamente. Assim como feito anteriormente, executou-se A1 e A2 10 vezes para a resolução de cada um dos

[4] P6.2 nada mais é do que uma cópia de P6. Aqui este problema é chamado de P6.2 com o fim de reforçar sua relação com os problemas P6.3 a P6.6.

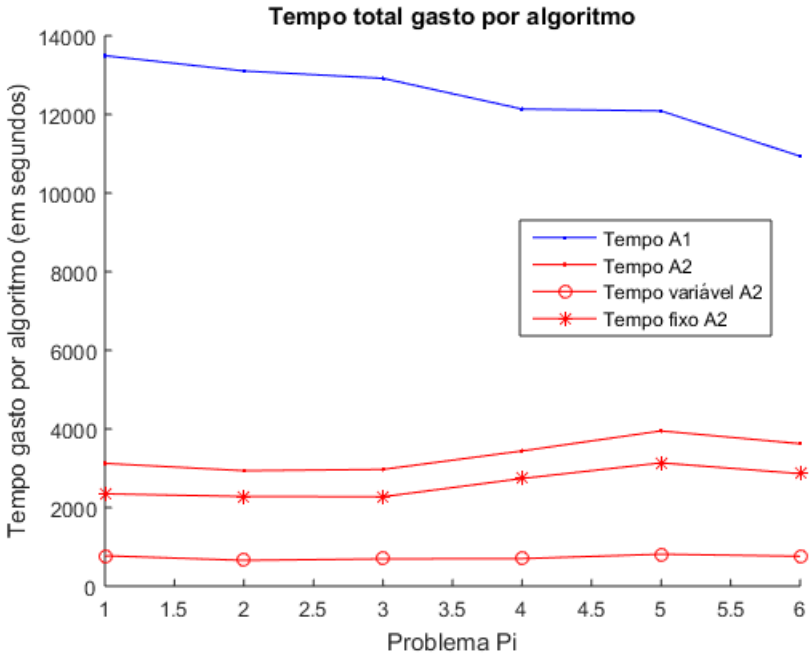


Figura 35 – Tempo gasto pelos algoritmos.

problemas P6.2 a P6.6. Os resultados obtidos por cada simulação podem ser visualizados nas figuras (36) e (37).

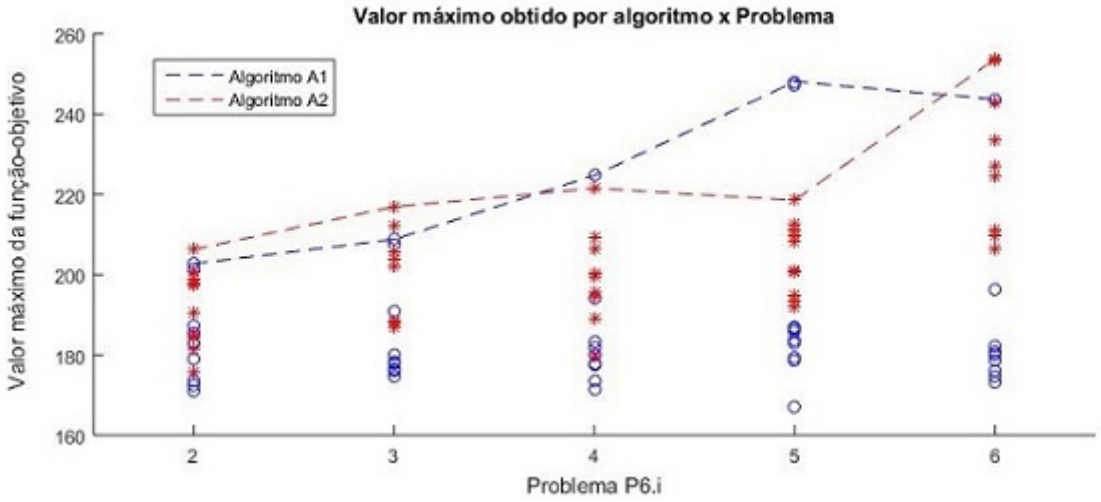


Figura 36 – Aptidão máxima dos indivíduos de cada simulação dos problemas P6.2 a P6.6.

Como esperado, o tempo gasto por A1 e A2 crescem à medida que o número de cidades de um problema aumenta. Porém, A1 sofre uma maior variação do que A2, mostrando-se mais sensível ao número de cidades de um problema. Além disso, o desem-

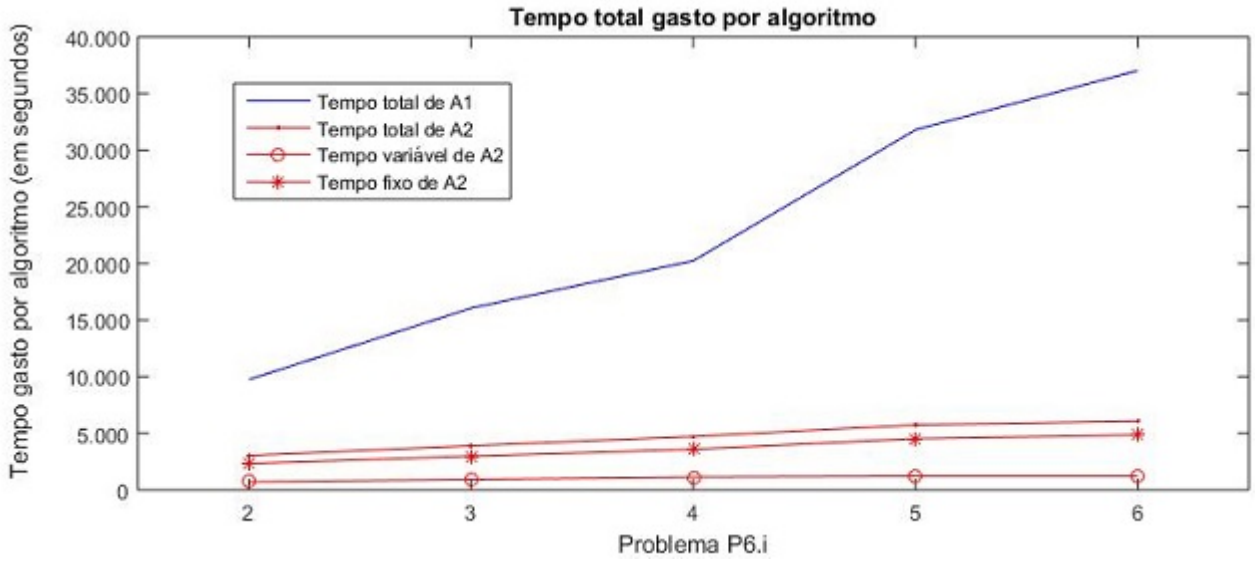


Figura 37 – Tempo gasto pelos algoritmos por simulação de P6.2 a P6.6.

penho dos algoritmos não se mostra mais tão distante se comparado à simulações dos problemas P1 a P6 anteriores, de modo que A1 chega a encontrar uma melhor solução que A2 para o problema P6.5. Pode-se notar também que as aptidões dos indivíduos selecionados por A1 estão mais dispersas do que A2.

Por fim, tome o problema P6.6. O indivíduo de maior aptidão encontrado para este problema nas simulações anteriores possui função-objetivo valendo 253,7001. Esta solução foi encontrada diversas vezes na 9ª simulação do algoritmo, aparecendo diversas vezes nas gerações 31,34, 37 a 39 e 43 a 47 (de um total de 56 gerações)[5]. Denotando esta solução como $(\bar{x}_0, \bar{y}_0)$, defina o conjunto $C_N = \{(\bar{x}_1, \bar{y}_1), ..., (\bar{x}_N, \bar{y}_N)\}$, onde a distância entre $(\bar{x}_i, \bar{y}_i)$ a $(\bar{x}_{i+1}, \bar{y}_{i+1})$ é unitária $\forall i = 0, ..., N-1$. C é então um *caminho* de tamanho N com início em $(\bar{x}_0, \bar{y}_0)$.

Foram gerados aleatoriamente 300 caminhos de tamanho N com início em $(\bar{x}_0, \bar{y}_0)$, de modo a estudar a variação da aptidão de soluções a partir de um suposto "ótimo"[6]. A figura (38) mostra um *boxplot* que indica a variação da aptidão de indivíduos à medida que sua distância à solução ótima de referência é aumentada. Nestas figuras também é mostrada a aptidão de referência (na linha azul, com valor 253,7001) e a mediana das

[5] Esta solução não é encontrada a partir da geração 48, de modo que ela não foi selecionada pelo operador de seleção natural escolhido.

[6] Note que $(\bar{x}_0, \bar{y}_0)$ não é necessariamente o ótimo de P6.6, mas foi escolhido como ponto de partida por ser o maior valor obtido dentre as simulações feitas.

aptidões de todos os indivíduos (linha verde) à medida que a distância aumenta.

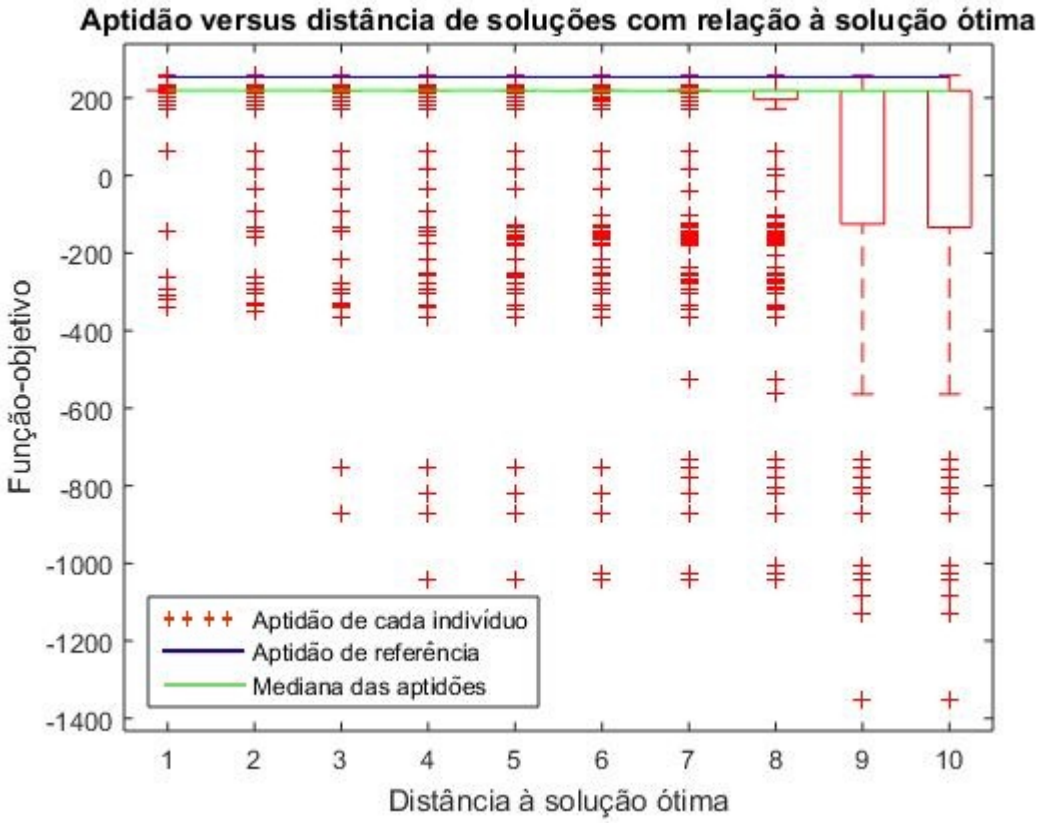


Figura 38 – Variação da aptidão de soluções à medida que se distanciam de $(\bar{x}_0, \bar{y}_0)$.

A figura (39) mostra um *zoom* feito na figura (38). Nesta figura é possível ver que há alguns indivíduos com aptidão superior à de $(\bar{x}_0, \bar{y}_0)$.

Como pode-se observar, a maior parte das soluções possui aptidão menor que a referência, mas ainda há soluções que melhoraram quando se afastaram de $(\bar{x}_0, \bar{y}_0)$, e como esperado, as aptidões das 300 soluções tendem (com exceções) a decrescer à medida que se afastam de $(\bar{x}_0, \bar{y}_0)$, já que esta solução é um mínimo local encontrado pelas simulações anteriores. Além disso, quase não houve variação na mediana das aptidões à medida que as 300 soluções se afastam de $(\bar{x}_0, \bar{y}_0)$.

Os resultados obtidos nesta seção serão discutidos mais a fundo na próxima seção.

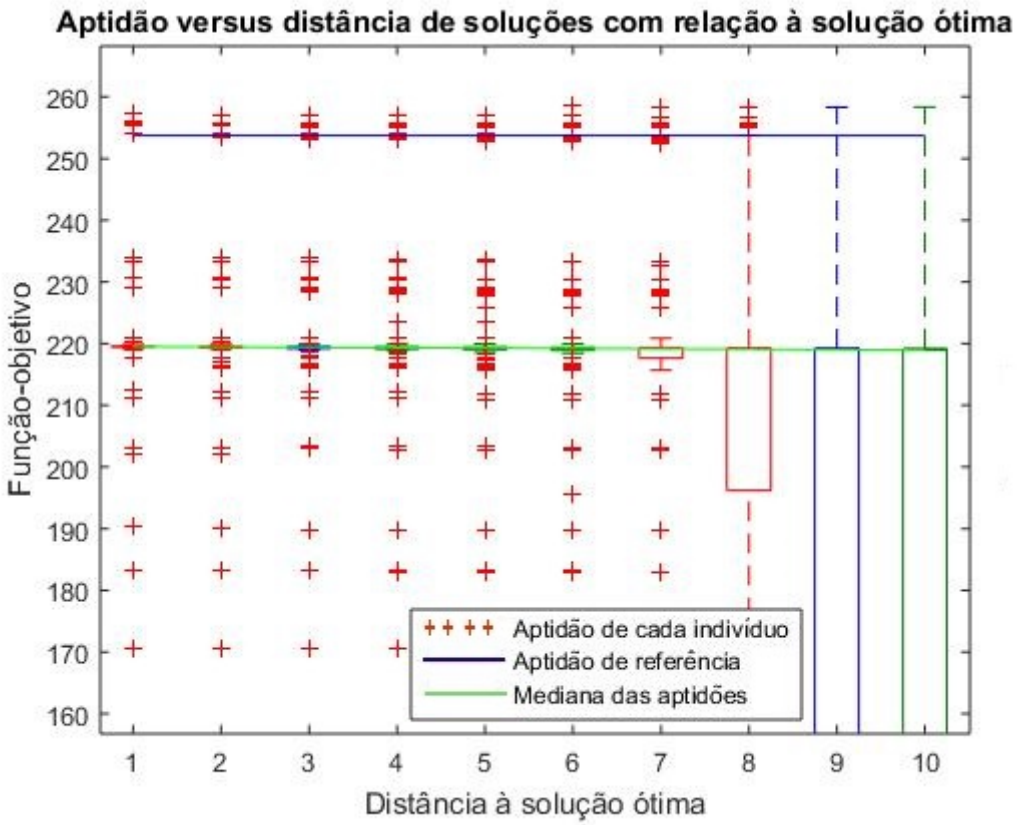


Figura 39 – Zoom feito na figura (38).

6.3 Discussão

Verificou-se de forma empírica que a simplificação de problemas em clusters apresenta um aumento do desempenho do algoritmo genético. Ao aproximar os centros geométricos de cada cluster é observado um aumento de eficiência do algoritmo A1 com relação a A2 se considerados apenas os valores da função-objetivo encontrados, mas curiosamente A2 continua obtendo melhores valores que A1, como pode ser observado pelas figuras (33) e (34). Vale lembrar que estas conclusões foram feitas observando o comportamento dos algoritmos via simulações sobre os problemas 1 a 6, de modo que estes resultados indicam uma melhoria na resolução de outros TTP1s via simplificação por clusters, mas não é possível afirmar que a superioridade de A2 sobre A1 prevalece em outros problemas.

Além disso, a figura (35) mostra a superioridade de A2 sobre A1 quando são feitas várias simulações. Observando esta figura, tem-se que A2 tem tempo fixo de aproximadamente 2100 segundos, mas cada simulação demora menos de 100 segundos para

ser executada após os cálculos dos caminhos entre cidades representantes. Assim, uma simulação de A2 demora cerca de 2100 segundos, enquanto uma única execução de A1 demora cerca de 1350s para P1. Porém, duas execuções de A2 para resolver P1 demoram cerca de 2200 segundos enquanto duas execuções de A1 demoram aproximadamente 2700 segundos. E como ambos os algoritmos podem retornar valores diversos após sua execução sobre um mesmo problema (como pode ser visto na figura (33)), então é necessária a execução sucessiva do mesmo algoritmo sobre um mesmo problema de modo a se obter melhores resultados.

Também, pode-se observar na figura (37) a diferença entre os tempos gastos pelos algoritmos quando o número de cidades cresce, mantendo-se a estrutura dos clusters de um problema de referência (neste caso, P6), onde A1, comparado a A2, demonstra ter um tempo de execução mais sensível à variação do tamanho do problema. Na figura (36) observa-se que proximidade entre as soluções encontradas pelos dois algoritmos se mantem, de modo que A1 encontra soluções com aptidões mais distantes entre si, enquanto A2 demonstra encontrar soluções de aptidão mais próxima. Isto se explica pelo fato de A1 fazer buscas menos restritas, enquanto A2 deve seguir a estrutura de seus clusters.

Finalmente, foram gerados 300 caminhos (de tamanho 10) a partir de uma dada solução $(\bar{x}_0, \bar{y}_0)$ de referência (obtida nas simulações de P6.6). Nas figuras (38) e (39) são mostrados os resultados obtidos, indicando que a métrica escolhida tende a gerar soluções de aptidão menor, se tomado como ponto de partida uma solução "ótima". Como a solução de referência não é um ótimo global, também foram obtidas algumas soluções de maior aptidão, mas como esta é um ótimo local, a tendência é que a função-objetivo de uma solução diminua à medida que sua distância à referência aumente.

No próximo capítulo será feita a conclusão deste estudo, recapitulando tudo o que foi feito de modo a expor as principais contribuições feitas, bem como traçar os próximos passos a serem tomados na continuação deste estudo.

7 Conclusão

Neste capítulo é feita a conclusão deste estudo, sendo organizado da seguinte forma:

Na seção seção 7.1 é feita a recapitulação do problema, e das principais dificuldades que ele apresentou inicialmente. Também é feita uma análise da metodologia utilizada, expondo novamente seus objetivos, e o que foi feito neste estudo.

Na seção seção 7.2 são destacados os principais resultados obtidos neste estudo, resumindo o que foi exposto no Capítulo 6. Nesta seção também são destacadas as principais contribuições deste estudo.

Na seção seção 7.3 são debatidas possíveis melhorias que podem ser feitas, explorando novas abordagens ao problema estudado, bem como novas metodologias de estudo.

7.1 Análise Metodológica

Este estudo é iniciado pela enunciação do Problema do Caixeiro Viajante, e do Problema da Mochila, de modo a construir um novo problema: O Problema do Ladrão Viajante.

Tendo como motivação a melhor modelagem de problemas reais complexos, é enunciada uma possível junção de dois problemas clássicos de otimização, criando assim o TTP1. Nesta fusão de problemas, são exploradas características de ambos os problemas que são muito úteis para modelar problemas de logística de transportes por exemplo, mas este problema não necessariamente se limita a modelar este tipo de problema, podendo haver outros que ainda não foram explorados.

Este novo problema se mostrou diferente de seus predecessores não apenas em sua complexidade, de modo que a interconexão feita entre os dois problemas de origem permite que a fusão de soluções ótimas nestes problemas não ser uma solução ótima do TTP1.

Para abordar este problema novo, foi feito um algoritmo genético denominado A1. Para a criação de tal algoritmo foi necessário o desenvolvimento de operadores específicos a este problema, devido à natureza da representação de seus cromossomos.

Como A1 não explorava quaisquer características que o TTP1 escolhido possa ter, foram estudadas possíveis simplificações do problema que explorem suas características, de modo a tornar a busca por soluções ótimas mais rápida, e que as soluções obtidas fossem mais aptas. Deste estudo foi escolhida a abordagem do problema utilizando clusters.

Nesta nova abordagem, agrupam-se cidades que possuam os mesmos itens disponíveis (de preferência cidades próximas espacialmente), de modo que quaisquer cidades fora de um determinado grupo não pode possuir algum item disponível naquele grupo. Além disso, cidades que não possuem itens disponíveis podem fazer parte de quaisquer grupos, de modo que o ideal seja colocar estas cidades no grupo que possua a cidade (que tenha objetos) mais próxima.

Assim, é explorada a possibilidade de fazer buscas mais direcionadas, de modo a se procurar caminhos que ligam clusters. Esta simplficação tem o preço de se precisar procurar caminhos mínimos dentro de cada cluster, interligando suas *cidades representantes*. Desta forma, surge o algoritmo A2, que nada mais é do que A1 executado sobre a simplificação do TTP1 estudado.

7.2 Resultados Obtidos

Após desenvolver A1 e A2, foram feitos testes em classes específicas de problemas de modo a avaliar de forma empírica qual a diferença de desempenho dos dois algoritmos para diversas configurações de problemas.

Inicialmente, é definido um problema P6 onde suas cidades estão muito bem agrupadas em 9 clusters, sendo um formado apenas pela cidade inicial do trajeto do ladrão. Neste problema, há 8 objetos disponíveis e cada cidade do problema possui um, ou nenhum objeto. As cidades que podem ter o objeto 1 por exemplo, estão agrupadas numa mesma região delimitada por um círculo de raio $\frac{\pi}{2}$, de modo que esta região delimita um cluster que contém este objeto. O mesmo vale para os demais objetos, de modo que é esperado que neste problema o algoritmo A2 tenha melhor desempenho, visto que o algoritmo A1 irá selecionar (diversas vezes) em suas soluções arestas que ligam cidades de clusters diferentes, ocasionando em caminhos de "vai-e-vem" que aumentam o tempo gasto para efetuar o trajeto desnecessariamente, reduzindo o valor das funções-objetivo de suas soluções.

Como esperado, A2 encontrou melhores valores que A1. Além disso, como os valores

retornados por ambos os algoritmos varia, é necessária a execução do mesmo algoritmo sobre um problema diversas vezes, de modo a se obter melhores soluções, e devido a esta característica A2 também se mostra mais rápido que A1, sendo mais rápido que A1 a partir da segunda execução.

O mesmo tipo de experimento foi feito com outros 5 problemas, que nada mais são do que o mesmo tipo de problema, mas com os clusters menos evidentes. A partir de P5 os clusters são aproximados espacialmente, ficando cada vez menos evidentes, de modo que em P1, todas as cidades se encontram arbitrariamente espalhadas em torno da cidade inicial.

Porém, A2 se mostrou superior que A1 em todos os demais problemas testados, incluindo P1. O tempo de execução dos dois algoritmos não variou muito, mas a aptidão das soluções encontradas por A1 cresceu até ficarem próximas (mas em sua maioria, ainda inferiores) às encontradas por A2.

Este resultado curioso não foi esperado antes de se fazer tais simulações, mas ainda pode ser explicado pelo fato de A1 ser um algoritmo que faz buscas "cegas". Devido à complexidade deste problema, de modo que pequenas mudanças em soluções podendo acarretar em grandes variações de suas aptidões, é compreensível que qualquer "ajuda" dada ao algoritmo - por meio de simplificações do problema inicial por exemplo - possa melhorar seu desempenho, mesmo que estas simplificações limite o espaço de busca do algoritmo (que é o caso da simplificação por clusters).

Além disso, foram feitas simulações para comparação entre os algoritmos à medida que o número de cidades do problema aumentam (mantidas suas outras características). Como esperado, os algoritmos possuem tempo de execução crescente à medida que o número de cidades aumenta, e A1 se mostra mais sensível à estas variações. Da mesma forma, é observado que as aptidões dos indivíduos encontrados pelos algoritmos permanece próxima em problemas com cidades aglomeradas e sem clusters "óbvios", de modo que A1 gera soluções com aptidões mais dispersas, enquanto A2 gera soluções mais concentradas.

Por fim, foi feita uma simulação para analisar a variação da função-objetivo de soluções à medida que estas distam de uma dada solução "ótima" de referência (um ótimo local obtido em simulações périvas). Como esperado, as soluções tendem a ter uma aptidão menor que a solução de referência à medida que se distanciam desta. Há também algumas

exceções (soluções que melhoraram ao se distanciarem da referência) que se devem ao fato de a solução de referência não ser um ótimo global.

Assim, este estudo contribui não apenas com um algoritmo genético para a busca de soluções do TTP1 (com operadores específicos para este problema), mas principalmente com a definição de clusters para este problema, suas propriedades, e seu impacto (estudado empiricamente) na busca por soluções deste problema complexo.

7.3 Trabalhos Futuros

O estudo desta classe de problemas ainda tem um longo caminho pela frente. Em trabalhos futuros ainda é possível estudar outras propriedades que o TTP1 possa ter, e que ainda não foram exploradas. Desvendar estas possíveis propriedades, e descobrir como utilizar tais propriedades como auxílio na busca por soluções ótimas pode ser um bom "próximo passo".

Além disso, o agrupamento de cidades ainda pode ser mais estudado. Novas formas de se explorar o agrupamento de cidades pelos clusters definidos neste estudo, ou mesmo definir outros tipos de agrupamento ainda são uma possibilidade.

Da mesma forma, outras possíveis junções entre o TSP e o KP, definindo novas restrições e variáveis que interconectam suas principais características ainda são opções de estudos futuros, de modo que diferentes junções destes problemas podem se mostrar mais eficientes para modelar diferentes problemas (além de terem novas propriedades a serem descobertas).

Por fim, um estudo mais conceitual sobre a diferença de desempenho entre os algoritmos A1 e A2, de modo a se investigar de forma mais profunda quais foram as principais causas desta diferença de desempenho também são tópicos para possíveis trabalhos futuros.

Assim, este estudo é concluído, não apenas avaliando o que foi feito, mas também traçando os próximos passos a serem seguidos no estudo deste problema complexo, mas também desafiador e interessante.

Referências

1 GASPAR-CUNHA, A.; TAKAHASHI, R. H. C.; ANTUNES, C. H. *MANUAL DE COMPUTAÇÃO EVOLUTIVA E META-HEURÍSTICAS*. [S.l.: s.n.], 2013. Citado na página 19.

2 CARRANO, E. G.; TAKAHASHI, R. H. C. On a vector space representation in genetic algorithm for sensor scheduling in wireless sensor networks. In: *Evolutionary Computation*. [S.l.: s.n.], 2013. Citado 4 vezes nas páginas 19, 31, 33 e 53.

3 CARRANO, E. G. et al. Subpermutation based evolutionary multiobjective algorithm for load restorarion in power distribution networks. In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*. [S.l.: s.n.], 2014. Citado na página 19.

4 HELSGAUN, K. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, p. 106–130, 1999. Citado na página 19.

5 HERNANDO, L. et al. A study on the complexity of tsp instances under the 2-exchange neighbor system. 2011. Citado na página 19.

6 ILAVARASI, K.; JOSEPH, K. S. Variants of travelling salesman problem: A survey. *ICICES2014*, 2014. Citado 2 vezes nas páginas 19 e 64.

7 IBARRA, O. H.; KIM, C. E. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the Association for Computing Machinery*, v. 22, n. 4, p. 463–468, oct 1975. Citado na página 19.

8 GAVISH, B.; PIRKUL, H. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. In: *MATHEMATICAL PROGRAMMING*. [S.l.: s.n.], 1985. v. 31, p. 78–105. Citado na página 19.

9 CHU, P.; BEASLEY, J. A genetic algorithm for the multidimensional knapsack problem. In: *JOURNAL OF HEURISTICS*. [S.l.]: KLUWER ACADEMIC PUBLISHERS, 1998. v. 4, n. 63-86. Citado 2 vezes nas páginas 19 e 20.

10 JASZKIEWICZ, A. On the performance of multiple-objective genetic local search on the 0/1 knapsack problem – a comparative experiment. In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*. [S.l.: s.n.], 2002. v. 6, n. 4. Citado na página 19.

11 NOBIBON, F. T.; LEUS, R. Complexity and approximability results for robust knapsack problems. *Proceedings of the 2010 IEEE IEEM*, 2010. Citado na página 19.

12 DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: Optimization by a colony of cooperating agents. In: *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B CYBERNETICS*. [S.l.: s.n.], 1996. v. 26, n. 1. Citado na página 19.

13 CARRANO, E. G.; TAKAHASHI, R. H. C. Nonlinear network optimization - an embedding vector space approach. In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*. [S.l.: s.n.], 2010. v. 14, n. 2. Citado na página 20.

14 BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. In: *ACM Computing Surveys*. [S.l.: s.n.], 2003. v. 35, n. 3, p. 268–308. Citado na página 20.

15 BONYADI, M. R.; MICHALEWICZ, Z.; BARONE, L. The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. *IEEE Congress on Evolutionary Computation June 20-23*, 2013. Citado 6 vezes nas páginas 20, 23, 25, 26, 27 e 29.

16 MOHAMED, E. Y.; BELAïD, A. Cosolver2b: An efficient local search heuristic for the travelling thief problem. may 2016. Citado na página 20.

17 MLADENOVIC, N.; HANSEN, P. Variable neighborhood search. In: *Computers Ops. Res.* [S.l.: s.n.], 1997. v. 24, n. 11, p. 1097–1100. Citado na página 20.

18 XU, R.; II, D. W. Survey of clustering algorithms. In: *IEEE TRANSACTIONS ON NEURAL NETWORKS*. [S.l.: s.n.], 2005. v. 16, n. 3. Citado na página 20.