

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RIO GRANDE DO
NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

Tarso Nunes Aires

**ANÁLISE DE DESEMPENHO DE ALGORITMOS DE BUSCA NA RESOLUÇÃO DO
PROBLEMA DO CAIXEIRO VIAJANTE**

**NATAL/RN
2009**

TARSO NUNES AIRES

**ANÁLISE DE DESEMPENHO DE ALGORITMOS DE BUSCA NA RESOLUÇÃO DO
PROBLEMA DO CAIXEIRO VIAJANTE**

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador:

Alexandre Luiz Galvão Damasceno, Me.

**NATAL/RN
2009**

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RIO GRANDE DO
NORTE
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

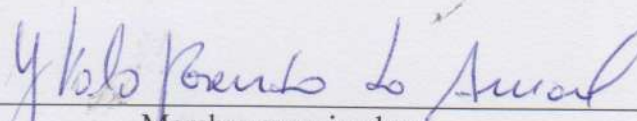
Tarso Nunes Aires

**ANÁLISE DE DESEMPENHO DE ALGORITMOS DE BUSCA NA RESOLUÇÃO DO
PROBLEMA DO CAIXEIRO VIAJANTE**

Monografia apresentada e aprovada em _____ de _____ de _____,
pela banca examinadora composta pelos seguintes membros:

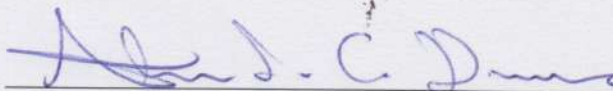


Alexandre Luiz Galvão Damasceno, Me.
Orientador



Membro examinador

YTAIO ROSENDO DO AMARAL



Membro examinador

Natal/RN, _____ de _____ de _____.

Aos meus pais, por toda a educação proporcionada,
sem a qual não seria possível concluir este trabalho.

À minha esposa Cristiane, pelo amor e apoio
constantes.

À minha filha Letícia, que rege todas as minhas
atitudes e planos.

AGRADECIMENTOS

Este trabalho foi construído com o apoio de várias pessoas e instituições. Por isso, venho agradecer a todos com as mais sinceras palavras, esperando ter atingido os fins pretendidos neste empenho conjunto e atendido as expectativas de todos que contribuíram para sua construção.

Assim, agradeço a todos os meus professores, pelos ensinamentos ministrados e pelo estímulo aos desafios profissionais e acadêmicos que deles recebi.

Aos meus professores revisores e orientadores, em especial o Prof. Alexandre Luiz Galvão Damasceno, por sua paciência e contribuição em tornar este trabalho melhor; o Prof. Ytalo Rosendo do Amaral, pelas orientações e dicas valiosas; e a Prof^a Joseane Pinheiro, que espontaneamente se envolveu com a revisão deste trabalho e fez observações valiosíssimas acerca de seu conteúdo.

À Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, particularmente nas pessoas de seus Diretores, Professores, Coordenadores e Funcionários, pela presteza e suporte que me ofereceram para a realização e conclusão desta oportunidade produtiva e valorosa que é a produção de conhecimento.

Aos alunos da turma de 2006 de Bacharelado em Sistemas de Informação, uma das turmas mais soberbas e prestativas que já tive a oportunidade de estudar.

E a todos que, de alguma forma, contribuíram na elaboração deste trabalho.

SUMÁRIO

LISTA DE FIGURAS	6
LISTA DE TABELAS.....	7
LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS	8
1. INTRODUÇÃO	11
2. OBJETIVOS	12
2.1. Objetivo Geral.....	12
2.2. Objetivos Específicos.....	12
3. JUSTIFICATIVA.....	13
4. REFERENCIAL TEÓRICO.....	14
5. METODOLOGIA	30
6. ANÁLISE DE RESULTADOS	33
7. CONCLUSÃO	51
7.1. Melhorias e Pontos de Partida.....	52
8. BIBLIOGRAFIA.....	53

LISTA DE FIGURAS

FIGURA 1 – Instância do Problema do Caixeiro Viajante com quatro cidades.....	17
FIGURA 2 – Topologia de espaço de estados	23
FIGURA 3 – Tabela comparativa de algoritmos de busca	28
FIGURA 4 – Tempo médio de execução em função do número de cidades	48
FIGURA 5 – Valores médios das soluções em função do número de cidades	49
FIGURA 6 – Erro médio relativo à solução ótima em função da quantidade de cidades...	50

LISTA DE TABELAS

TABELA 1 – Lista das 64 instâncias do PCV selecionadas	31
TABELA 2 – Configuração do computador utilizado para execução dos algoritmos.....	32
TABELA 3 – Parâmetros de execução dos algoritmos e seus valores	32
TABELA 4 – Simulated Annealing, 1ª execução (a280 a pcb442)	34
TABELA 5 – Simulated Annealing, 1ª execução (pr1002 a vm1748)	35
TABELA 6 – Simulated Annealing, 2ª execução (a280 a pcb442)	36
TABELA 7 – Simulated Annealing, 2ª execução (pr1002 a vm1748)	37
TABELA 8 – Simulated Annealing, 3ª execução (a280 a pcb442)	38
TABELA 9 – Simulated Annealing, 3ª execução (pr1002 a vm1748)	39
TABELA 10 – Algoritmos Genéticos, 1ª execução (a280 a pcb442).....	40
TABELA 11 – Algoritmos Genéticos, 1ª execução (pr1002 a vm1748).....	41
TABELA 12 – Algoritmos Genéticos, 2ª execução (a280 a pcb442).....	42
TABELA 13 – Algoritmos Genéticos, 2ª execução (pr1002 a vm1748).....	43
TABELA 14 – Algoritmos Genéticos, 3ª execução (a280 a pcb442).....	44
TABELA 15 – Algoritmos Genéticos, 3ª execução (pr1002 a vm1748).....	45

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

AG = Algoritmos Genéticos

PCV = Problema do Caixeiro Viajante

PRV = Problema do Roteamento de Veículos

SA = Simulated Annealing

TSPLIB = Traveling Salesman Problem Library

RESUMO

O objetivo deste trabalho é realizar um estudo que permita apresentar dados relativos ao desempenho dos algoritmos Simulated Annealing e Algoritmos Genéticos na resolução do Problema do Caixeiro Viajante. Os resultados desse estudo poderão ser usados como base empírica para tomar decisões de implementação durante o desenvolvimento de um software que precise determinar a melhor rota entre dois pontos distintos. Inicialmente fez-se um estudo bibliográfico do problema de pesquisa, e em seguida realizou-se um levantamento das instâncias do problema a serem resolvidas. Construiu-se então o software que executa os algoritmos, e após isso foram colhidos dados em forma de tabelas e gráficos que deram respaldo para a construção da análise. Com base nos critérios de análise selecionados, que são complexidade de desenvolvimento do algoritmo, desempenho e eficiência, considera-se que o Simulated Annealing seja mais adequado que os Algoritmos Genéticos para resolver instâncias do Problema do Caixeiro Viajante, especialmente quando a quantidade de cidades por instância ultrapassa mil.

Palavras-chave:

Algoritmos. Problema do Caixeiro Viajante. Simulated Annealing. Algoritmos Genéticos.

ABSTRACT

The objective of this paper is to conduct a study in order to display data on the performance of Simulated Annealing and Genetic Algorithms in solving the Traveling Salesman Problem. The results of this study could be used as empirical basis for making implementation decisions during the development of software that need to determine the best route between two different points. Initially we carried out a bibliographic review of the research problem, and then we carried out a survey of the instances of the problem to be solved. Then a software that runs the algorithms was builded, and then the data were collected in form of tables and charts that give support for the analysis construction. Based on the criteria selected for analysis, which are the complexity of the algorithm development, performance and efficiency, it is considerate that the Simulated Annealing is more appropriate than Genetic Algorithms to solve TSP instances, especially when the number of cities exceeds a thousand.

Keywords:

Algorithms. Traveling Salesman Problem. Simulated Annealing. Genetic Algorithms.

1. INTRODUÇÃO

Há tempos o homem utiliza-se da matemática e da computação para garantir seu progresso e sua sobrevivência, aplicando-as na resolução dos mais variados problemas em áreas igualmente diversas. Praticamente todas as ciências e disciplinas dos dias atuais munem-se dos princípios matemáticos e das ferramentas computacionais para comprovação e formulação de teorias, para que estas possam ser aplicadas na obtenção de resultados práticos.

A Ciência da Computação engloba áreas de pesquisa que foram criadas com o objetivo de abordar problemas de elevada complexidade. Uma dessas áreas que se encontra em plena expansão atualmente é a Inteligência Artificial – dedicada a buscar meios computacionais com capacidade de simular a inteligência humana. Seu desenvolvimento possibilitou a criação de meios utilizados tanto para resolver questões até então insolúveis quanto para aprimorar as soluções já existentes, muitas vezes aplicando-se uma ótica diferente das que vinham sendo utilizadas.

Existe uma área da Inteligência Artificial que abrange o desenvolvimento de algoritmos utilizados para determinar soluções ótimas – ou próximas do ótimo – para diversos problemas. Esses algoritmos são chamados algoritmos de busca. Uma das questões que podem ser resolvidas através de algoritmos de busca é o Problema do Caixeiro Viajante (PCV), onde dado um conjunto de cidades interligadas entre si, precisa-se determinar um caminho que passe por todas elas uma única vez, partindo de uma cidade inicial e retornando a ela ao final do percurso.

A determinação de soluções para o PCV é bastante conveniente para empresas situadas nos ramos de transporte de cargas e de viagens aéreas, por exemplo. Isso porque, obviamente, quanto melhor o caminho encontrado para percorrer os destinos, menos custoso será o transporte. Muitos algoritmos de busca já foram empregados ao problema, e há muitos estudos realizados sobre o desempenho desses algoritmos quando utilizados para a resolução de diferentes instâncias do PCV. Porém, diante de variáveis como a quantidade de cidades a ser percorrida e a distância entre elas, há algum algoritmo de busca que seja mais equilibrado em termos de custo computacional e qualidade da solução? Para quais valores dessas variáveis os algoritmos devem ser empregados?

Este trabalho se propõe a realizar um estudo de um conjunto de algoritmos de busca empregados na resolução de diferentes instâncias do PCV, onde cada uma é uma variação do conjunto de variáveis que o define, listadas informalmente no parágrafo anterior.

2. OBJETIVOS

2.1. OBJETIVO GERAL

Realizar, através da construção de um software, um estudo que permita apresentar dados relativos ao desempenho de diversos algoritmos de busca na resolução de diferentes instâncias do Problema do Caixeiro Viajante.

2.1. OBJETIVOS ESPECÍFICOS

- Formular o Problema do Caixeiro Viajante e definir as variáveis que diferenciam suas instâncias;
- Desenvolver um software que modele o Problema do Caixeiro Viajante e que permita aplicar algoritmos de busca à solução de diferentes instâncias do problema;
- Utilizar o software para aplicar os algoritmos Simulated Annealing e Algoritmos Genéticos na busca de soluções para o Problema do Caixeiro Viajante;
- Utilizar os dados de saída do software para realizar uma análise comparativa de desempenho de cada algoritmo na resolução do problema;
- Apresentar pontos positivos e negativos das soluções empregadas, levando em consideração critérios como o custo computacional e a qualidade da solução encontrada para o problema.

3. JUSTIFICATIVA

No ramo de fornecimento e entrega de mercadorias, o custo final de um determinado produto sofre influência de vários fatores, dentre eles o quanto foi gasto com a sua distribuição. Já no ramo de viagens aéreas, o preço da passagem depende diretamente, dentre outros fatores, da distância a ser percorrida pela aeronave. O PCV está presente no dia a dia de várias empresas cujos nichos são transporte de cargas ou viagens, e resolvê-lo é fundamental para reduzir o percentual de gastos com esses transportes a um nível mais satisfatório, impactando em redução de preços nos produtos finais.

Porém, o PCV pode se apresentar de inúmeras formas, variando desde a quantidade de cidades a serem percorridas até a distância entre as mesmas, e da mesma maneira existem vários algoritmos potencialmente adequados para a busca de uma solução para o problema. Este trabalho visa apresentar dados empíricos que permitam chegar a uma conclusão quanto ao desempenho dos algoritmos e as instâncias do PCV para as quais eles se mostram mais adequados.

4. REFERENCIAL TEÓRICO

Nesta seção serão introduzidos os conceitos necessários para o entendimento do trabalho que será realizado. Inicialmente apresentam-se os significados de agentes, agentes inteligentes e agentes de resolução de problemas. Em seguida, disserta-se sobre problemas e formulação de problemas, para que então o problema central deste trabalho – o Problema do Caixeiro de Viajante – possa ser definido formalmente. Por fim, as estratégias de busca utilizadas para a sua resolução são apresentadas e discutidas.

4.1. AGENTES E AGENTES INTELIGENTES

Russel e Norvig (2003) introduziram os conceitos de agente, medida de desempenho e agente inteligente. Um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores, e de agir sobre esse ambiente por meio de atuadores. Ou seja, um agente, quando inserido em um ambiente, gera uma seqüência de ações de acordo com as percepções que recebe, e essa seqüência de ações faz o ambiente passar por uma seqüência de estados.

A medida de desempenho do agente pode ser definida como um critério para determinar o quão desejável é a seqüência de estados pelos quais o ambiente passou, em decorrência das ações do agente. Quantitativamente falando, a medida de desempenho de um agente é diretamente proporcional à quantidade de estados desejáveis do ambiente gerados pelas suas ações. Dessa forma, um agente inteligente ou racional deve executar, para cada seqüência de percepções possível, uma ação que venha a maximizar sua medida de desempenho.

4.2. AGENTES DE RESOLUÇÃO DE PROBLEMAS

Para Russel e Norvig (2003), um agente de resolução de problemas é um agente inteligente que procura maximizar sua medida de desempenho através da adoção de um

objetivo. Este, por sua vez, consiste no conjunto de estados do mundo que constituem a satisfação do problema ao qual o agente se propõe a resolver. Munido de um objetivo, um agente pode facilmente descartar os estados que não satisfazem o problema, simplificando o processo de tomada de decisão.

4.3. BUSCA E SOLUÇÃO DE PROBLEMAS

Um agente com várias opções imediatas de valor desconhecido pode decidir o que fazer examinando primeiro diferentes seqüências de ações possíveis que levam a estados de valor conhecido, e depois escolhendo a melhor seqüência. Esse processo de procurar por tal seqüência é chamado busca (Russel e Norvig, 2003, p. 62).

Um algoritmo de busca recebe um problema como entrada e retorna uma solução sob a forma de um caminho de ações, que quando executadas levam o agente do estado inicial até um estado objetivo.

4.4. FORMULAÇÃO DE PROBLEMAS

Russel e Norvig (2003) definiram uma técnica de formulação de problemas que consiste em enumerar os seguintes itens para o problema a ser formulado:

- O estado inicial em que o agente começa;
- Uma função sucessora que, dado um estado particular x , retorna um conjunto de pares ordenados (ação, sucessor) em que cada ação é uma das ações válidas no estado x e cada sucessor é um estado que pode ser alcançado a partir de x aplicando-se a ação;
- O conjunto de todos os estados acessíveis a partir do estado inicial, chamado de espaço de estados do problema e definidos com o auxílio do estado inicial e da função sucessora;
- O teste de objetivo, que determina se um dado estado é um estado objetivo;
- Uma função de custo de caminho que atribui um custo numérico a cada caminho e reflete a medida de desempenho do agente.

4.5. GRAFOS E CICLO HAMILTONIANO

Segundo Cormen *et al.* (2002), um grafo, informalmente falando, é representado como um conjunto de vértices (ou pontos) ligados por arestas. Essas arestas podem ser direcionadas ou não, e caso o sejam, são representadas por setas; caso contrário, utilizam-se retas. De maneira mais formal, um grafo é um par ordenado $G = (V, E)$ contendo um conjunto V de vértices ou nós e um conjunto E de arestas ou linhas, onde cada linha liga dois nós do conjunto V , distintos ou não. Ainda, um grafo é classificado como completo se cada par de vértices tem uma aresta correspondente que os conecta.

Cormen *et al.* (2002) definem um ciclo C de um grafo $G = (V, E)$ como sendo um conjunto de vértices $\{v_0, v_1, v_2, \dots, v_k\}$ pertencentes a V , com $v_0 = v_k$, que formam um caminho a ser percorrido pelas arestas de G . Ainda de acordo com os autores, C é classificado como hamiltoniano se contém cada vértice de G . Isso quer dizer que um ciclo hamiltoniano consiste em um caminho de um grafo G que é percorrido partindo de um vértice inicial v_0 , passa por todos os vértices de G apenas uma vez e retorna a v_0 .

4.6. O PROBLEMA DO CAIXEIRO VIAJANTE (PCV)

O PCV está intimamente relacionado ao ciclo hamiltoniano, apresentado na seção anterior. Nele, conforme Cormen *et al.* (2002), um vendedor deve visitar um número “ n ” de cidades, minimizando a distância necessária para percorrer o caminho que passa por todas elas apenas uma vez.

De maneira mais formal, o PCV pode ser modelado com um grafo ponderado completo $G = (V, E)$, onde $V = \{v_0, v_1, v_2, \dots, v_n\}$ é um conjunto de vértices e $E = \{(v_i, v_j); i \neq j\}$ é um conjunto de arcos. Cada vértice de V representa uma cidade a ser visitada e cada arco de E é uma estrada que liga uma cidade a outra. Sabendo-se que existe um custo inteiro $c(i, j)$ para viajar pelo arco (v_i, v_j) , o problema consiste em determinar o caminho de menor custo a ser percorrido pelo vendedor, onde o custo total é a soma dos custos individuais das arestas que compõem a viagem.

A figura 1 é uma representação gráfica de uma instância do PCV contendo quatro cidades. As arestas grifadas representam um ciclo hamiltoniano de custo mínimo igual a 7.

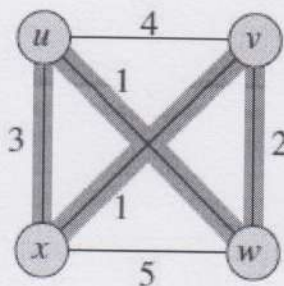


Figura 1: Uma instância do Problema do Caixeiro Viajante, modelada por um grafo com quatro vértices.
Fonte: Cormen *et al.*, 2002.

É importante mencionar que o PCV pode ser classificado em dois tipos, segundo Cormen *et al.* (2002). Dados dois vértices v_i e v_j pertencentes a V , se a distância de v_i para v_j for igual à distância de v_j para v_i , temos que o PCV é simétrico; caso contrário, o PCV é assimétrico. Todas as instâncias do PCV estudadas neste trabalho serão simétricas.

Sendo o agente de resolução deste problema o vendedor que realizará a visita a todas as cidades, pode-se formular o PCV utilizando a técnica de Russel e Norvig (2003), introduzida na seção anterior, da seguinte maneira:

- Estado inicial: a cidade de onde o vendedor partirá para percorrer seu caminho;
- Função sucessora: gera os estados válidos resultantes do deslocamento do vendedor de qualquer cidade a outra adjacente;
- Estados: cada uma das cidades contidas na malha geográfica é um estado válido do problema;
- Teste de objetivo: verifica se a cidade atualmente ocupada pelo vendedor é a cidade correspondente a seu ponto de partida;
- Custo do caminho: é a soma do custo de deslocamento de todas as cidades da rota tomada pelo vendedor, desde o ponto de partida até a cidade de destino.

Nas seções seguintes serão apresentadas diversas estratégias utilizadas para encontrar soluções para diferentes instâncias do PCV sob a forma de algoritmos de busca. Com o objetivo de dar um enfoque evolucionário a este referencial, alguns algoritmos de busca cega serão apresentados antes dos algoritmos heurísticos.

4.7. CRITÉRIOS DE AVALIAÇÃO DE ALGORITMOS DE BUSCA

Russel e Norvig (2003) afirmam que a saída de um algoritmo de busca consiste em uma solução ou falha, e definem quatro critérios para avaliação do desempenho desses algoritmos. São eles:

- Completeza: determina se o algoritmo garante que uma solução é encontrada quando ela existir;
- Otimização: determina se a solução encontrada é ótima, ou seja, a que possui menor custo de caminho;
- Complexidade de tempo: o tempo que a estratégia leva para encontrar uma solução, medido em termos do número de nós gerados durante a busca;
- Complexidade de espaço: quanta memória é necessária para executar a busca, medida em termos do número máximo de nós armazenado.

4.8. BUSCA CEGA

De acordo com Russell e Norvig (2003), a busca cega se caracteriza por não manter nenhuma informação adicional sobre estados além daquelas fornecidas na definição do problema. Ou seja, tudo o que elas podem realizar é gerar estados sucessores e distinguir um estado objetivo de um não-objetivo.

As estratégias de busca cega ilustradas neste trabalho serão a busca de custo uniforme e a busca em profundidade. Porém, para um bom entendimento da busca de custo uniforme, é necessário introduzir a busca em extensão.

4.8.1. BUSCA EM EXTENSÃO

A busca em extensão foi formulada inicialmente por Moore (1959) como uma estratégia de resolução de labirintos. Russell e Norvig (2003) a definem como “uma estratégia simples em que o nó raiz é expandido primeiro, em seguida todos os sucessores do nó raiz são expandidos, depois os sucessores desses nós e assim por diante”. Ou seja, em uma dada

árvore de busca, todos os nós em uma dada profundidade são expandidos antes dos nós do próximo nível da árvore.

Considerando que cada estado tem b sucessores, a raiz da árvore gera b nós no primeiro nível. Cada um deles também gera b nós no nível seguinte, e assim por diante. Se a solução estiver em uma profundidade d , no pior caso seriam expandidos todos os nós exceto o último do nível d , uma vez que o nó objetivo não é expandido. Assim, no nível $d + 1$ seriam gerados $b^{d+1} - b$ nós. Concluindo, o número total de nós da árvore de busca é indicado pela seguinte fórmula: $b + b^2 + b^3 + (...) + b^d + (b^{d+1} - b)$.

Russell e Norvig (2003) avaliam a busca em extensão usando os quatro critérios da seção 4.6 da seguinte forma:

- Completeza: dado que a quantidade máxima de sucessores de cada nó é limitada, se o nó objetivo mais raso da árvore estiver situado em uma profundidade finita, garante-se que o algoritmo sempre o encontrará. Nessas condições, a busca em extensão é completa;
- Otimização: se o custo do caminho crescer à medida que aumenta a profundidade do nó em expansão – isso ocorre, por exemplo, quando todas as ações têm o mesmo custo – a solução encontrada pelo algoritmo sempre será a melhor. Nessas condições, a busca em extensão é ótima;
- Complexidade de tempo e espaço: analisando a quantidade de nós gerados no pior caso, a complexidade de tempo e espaço da busca em extensão é $O(b^{d+1})$.

4.8.2. BUSCA DE CUSTO UNIFORME

Conforme seção anterior, a busca em extensão é ótima para uma situação muito particular. Russell e Norvig (2003) sugerem estender o algoritmo de forma que ele seja ótimo para qualquer função de custo de passo. Sendo $g(n)$ o custo do caminho da origem até o nó n , a busca de custo uniforme expande o nó com menor valor de $g(n)$. Se $g(n)$ for igual à profundidade de n , essa busca será idêntica à busca em extensão.

“A busca de custo uniforme não se importa com o número de passos que um caminho tem, mas apenas com o seu custo total”. (Russell e Norvig, 2003, p. 76).

Russell e Norvig (2003) avaliam a busca de custo uniforme usando os quatro critérios da seção 4.6 da seguinte forma:

- Completeza e otimização: dada uma constante positiva pequena ϵ , pode-se garantir que o algoritmo é completo e ótimo se o custo de cada passo for maior ou igual a ϵ ;
- Complexidade de tempo e espaço: a complexidade não pode ser caracterizada com facilidade em termos de b e d . Quando todos os custos de passos forem iguais, a complexidade será igual à busca em extensão.

4.8.3. BUSCA EM PROFUNDIDADE

Segundo Russell e Norvig (2003), “a busca em profundidade sempre expande o nó mais profundo na borda atual da árvore de busca” (a borda contém os nós que foram gerados, mas que ainda não foram explorados). A operação prossegue até que um nó sem sucessores seja encontrado – este é o nó mais profundo da árvore. Nesse ponto, a busca retorna ao antecessor mais próximo que tem sucessores ainda inexplorados.

Russell e Norvig (2003) avaliam a busca em profundidade usando os quatro critérios da seção 4.6 da seguinte forma:

- Completeza: a busca em profundidade não é completa, pois caso uma subárvore sendo expandida não tenha nó objetivo e sua profundidade seja ilimitada – por exemplo, quando a árvore possuir estados redundantes – a solução nunca será encontrada;
- Otimização: tomemos uma determinada árvore de busca que possui dois nós objetivos: um na subárvore esquerda, situado na profundidade 3, e outro na subárvore direita, situado na profundidade 1. Sabendo que a busca em profundidade encerra ao encontrar um nó objetivo, se a exploração iniciar-se pela subárvore esquerda, a solução encontrada não será a melhor solução. Portanto a busca em profundidade não é ótima;
- Complexidade de tempo: no pior caso, $O(b^d)$;
- Complexidade de espaço: a busca em profundidade exige o armazenamento de apenas $b(d + 1)$ nós. Portanto, a complexidade de espaço é $O(bd)$.

4.9. BUSCA HEURÍSTICA

A expressão “busca heurística” foi empregada inicialmente por Newell e Emst (1965) e Lin (1965) para designar estratégias de busca que utilizam, além da definição do problema, o conhecimento específico do próprio problema. Esses algoritmos selecionam o próximo nó para expansão tendo como base uma função de avaliação, cujo componente fundamental é uma função heurística $h(n)$ que toma um nó como entrada e retorna o custo estimado do caminho mais econômico do nó n até um nó objetivo.

De acordo com Russell e Norvig (2003), portanto, a função heurística $h(n)$ é uma função arbitrária específica de cada problema, com um caso particular: se n é um nó objetivo, então $h(n) = 0$.

A estratégia de busca heurística utilizada neste trabalho será a busca A^* .

4.9.1. BUSCA A^*

Os componentes de $f(x)$ na busca A^* , de acordo com Hart, Nilsson e Raphael (1972), são dois: a função $g(n)$, utilizada pela busca de custo uniforme que descreve o custo do caminho da origem até o nó n , e a função $h(n)$, que estima o menor custo estimado do caminho do nó n até um nó objetivo. Dessa forma, a função de avaliação $f(n)$ constitui o custo estimado da solução de menor custo passando por n , de maneira que o algoritmo buscará sempre expandir nós com o menor valor de $g(n) + h(n)$.

Russell e Norvig (2003) avaliam a busca A^* usando os quatro critérios da seção 4.6 da seguinte forma:

- **Completeza e Otimização:** Russel e Norvig (2003) ressaltam que é muito importante garantir que $h(n)$ nunca superestime o custo para alcançar o objetivo. Tal função é dita admissível se possuir essa característica e, dado que $g(x)$ sempre retorna o custo exato para se alcançar o estado n , conclui-se que, sendo $h(x)$ admissível, $f(x)$ jamais superestimar o custo verdadeiro de uma solução passando por n . Dechter e Pearl (1985) afirmam e provam, portanto, que a busca A^* será ao mesmo tempo completa e ótima se a função heurística satisfizer essa condição;

- Complexidade de tempo: função exponencial do comprimento da solução. Entretanto, quanto mais eficiente for a função heurística levantada para o problema, maior a economia de tempo em comparação com o uso de uma busca sem informação;
- Complexidade de espaço: $O(b^d)$, pois o algoritmo precisa manter todos os nós gerados na memória. Em geral, a busca A^* esgota o espaço antes mesmo de esgotar o tempo.

4.10. BUSCA LOCAL

Os algoritmos apresentados até agora possuem uma característica em comum: eles exploram espaços de busca de forma sistemática, mantendo um ou mais caminhos na memória e registrando as alternativas que foram exploradas e as que não foram exploradas em cada estado ao longo do caminho. Quando um estado objetivo é encontrado, o caminho do estado inicial até esse objetivo constitui uma solução para o problema.

Russel e Norvig (2003) afirmam que os algoritmos de busca local são os mais adequados para resolver problemas onde o caminho do nó inicial até o nó objetivo é irrelevante: esses algoritmos operam a partir de um único estado corrente e em geral se movem apenas para os estados adjacentes. Nesse processo, os caminhos seguidos pela busca não são armazenados, e como consequência direta desse fato, há um baixíssimo uso da memória. Dada a economia de espaço e de tempo, os algoritmos de busca local podem ser utilizados onde os sistemáticos são inadequados. Sendo assim, os algoritmos de busca local são úteis para a resolução de uma categoria de problemas definida como problemas de otimização puros, onde a meta é encontrar o melhor estado de acordo com uma função objetivo.

4.10.1. TOPOLOGIA DE ESPAÇO DE ESTADOS

Para entender a busca local, descobriremos que é muito útil considerar a topologia de espaço de estados. Uma topologia tem ao mesmo tempo “posição” (definida pelo estado) e “elevação” (definida pelo valor da função de custo da heurística ou função

objetivo). Se a elevação corresponder ao custo, o objetivo será encontrar o vale mais baixo – um mínimo global; se a elevação corresponder a uma função objetivo, então o objetivo será encontrar o pico mais alto – um máximo global (Russel e Norvig, 2003, p. 110)..

Ainda segundo os autores, um algoritmo de busca local é completo se ele sempre encontra um objetivo, caso ele exista; e ele será ótimo se encontrar sempre um mínimo ou máximo global.

A figura 2 ilustra uma topologia de espaço de estados onde a elevação, representada pelo eixo y, corresponde à função objetivo, e a posição, representada pelo eixo x, é definida pelo espaço de estados. O estado atual está assinalado no gráfico. A meta aqui é encontrar o máximo global; caso a elevação correspondesse ao custo, a meta seria encontrar o mínimo.

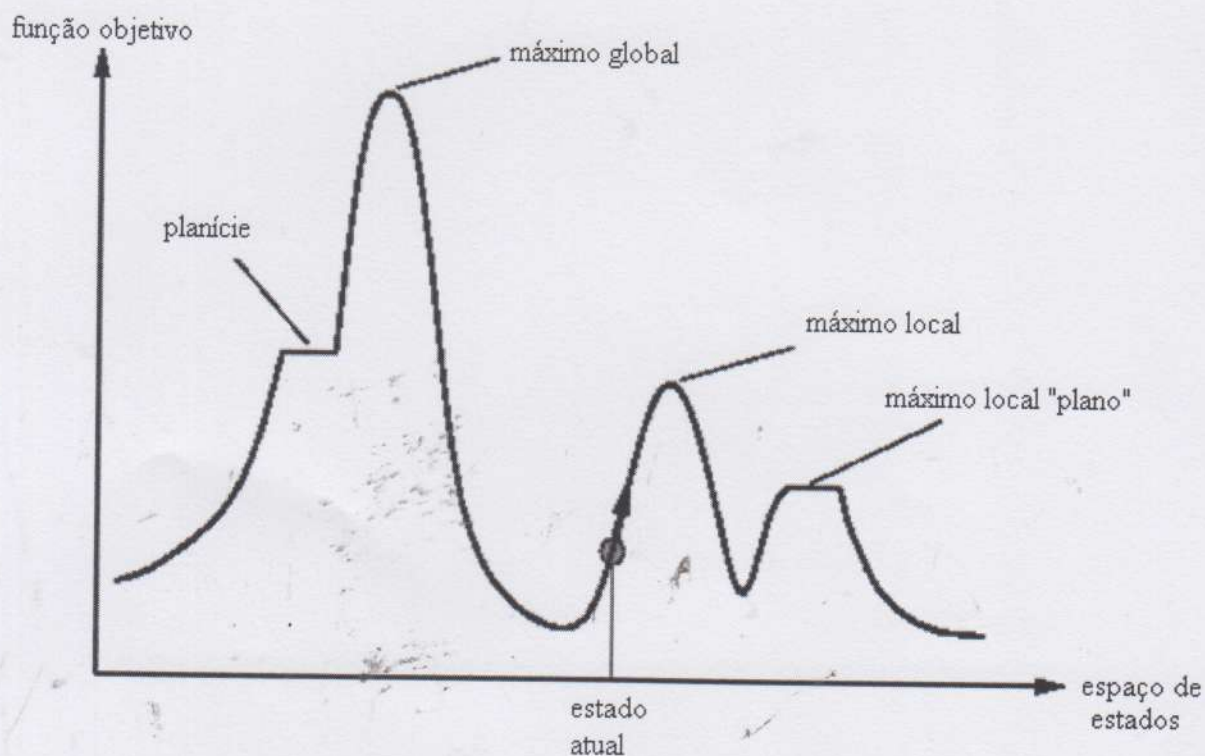


Figura 2: Topologia de espaço de estados unidimensional.
Fonte: Russel e Norvig, 2003.

Os algoritmos de busca local ilustrados nesse trabalho serão a Busca de Subida de Encosta, o Simulated Annealing e os Algoritmos Genéticos.

4.10.2. BUSCA DE SUBIDA DE ENCOSTA

Russel e Norvig (2003) definem a busca de subida de encosta como um algoritmo simples, onde um laço repetitivo se move de forma contínua no sentido do valor crescente. Quando o algoritmo encontra um “pico”, ou seja, um ponto da curva onde nenhum vizinho tem valor mais alto, ele termina. É considerada uma estratégia econômica, uma vez que não mantém uma árvore de busca, e sim uma estrutura de dados onde é registrado apenas o estado e o valor de sua função objetivo. Porém, ela apresenta uma grande limitação, como veremos a seguir.

Na figura 2 pudemos verificar a topologia de espaço de estados para compreender a busca local. Estão assinalados, além do estado atual, dois máximos – um global e outro local. Se iniciarmos a busca a partir do estado atual, logo encontraremos um máximo local; como não há nenhum estado vizinho com valor da função objetivo mais alto, o algoritmo será encerrado. E essa é a principal limitação da busca de subida de encosta: não há garantia de que o máximo encontrado seja o global, pois o processo fica paralisado em máximos locais.

Russel e Norvig (2003) apresentam uma abordagem que pode apaziguar essa situação, chamada subida de encosta com reinício aleatório. Um estado inicial é gerado ao acaso e o algoritmo é executado até que um estado objetivo seja encontrado. Após isso, outro estado é gerado e o algoritmo é executado novamente, e o processo continua até que uma determinada quantidade de reinícios ocorra. Os autores mostram que o simples fato de um estado objetivo ser eventualmente gerado confere à estratégia alta probabilidade de completudeza.

4.10.3. SIMULATED ANNEALING

Segundo Russel e Norvig (2003), há um processo em metalurgia denominado têmpera, que é utilizado para obtenção de estados de baixa energia num sólido e é composto de duas etapas. Na primeira, o sólido – por exemplo, metal ou vidro – tem sua temperatura aumentada para um valor máximo, na qual ocorre sua fusão. Na segunda, é realizado um resfriamento gradual ou arrefecimento do sólido até que ocorra sua solidificação. O resultado é que os átomos do material inicialmente ganham energia para se movimentarem livremente e, após o arrefecimento, se organizam numa configuração com menor energia interna.

Esse processo metalúrgico é utilizado como metáfora para o simulated annealing, ou têmpera simulada. O algoritmo parte de uma solução inicial qualquer e a substitui por uma solução adjacente no espaço de soluções, escolhida aleatoriamente de acordo com a função

objetivo e com uma variável T , que representa a temperatura. A componente aleatória que será incluída na próxima solução escolhida é diretamente proporcional a T , que por sua vez tem seu valor decrementado à medida que o algoritmo progride, provocando sua convergência para uma solução ótima, necessariamente local.

Conforme descrito inicialmente por Kirkpatrick, Gelatt e Vecchi (1983), o procedimento principal do algoritmo consiste em um laço que gera um único vizinho S' da solução corrente S a cada iteração. Após isso, calculam-se os valores da função objetivo para S' e para S – respectivamente, $f(S')$ e $f(S)$ – e então a variação da função objetivo $\Delta f = f(S') - f(S)$ é testada de acordo com as seguintes situações:

- Se $\Delta f < 0$, ou seja, se $f(S') < f(S)$, temos que S' é melhor do que S . O algoritmo adota S' como a nova solução atual;
- Se $\Delta f > 0$, ou seja, se $f(S') > f(S)$, temos que S' é pior do que S . Nesse caso, é necessário calcular a probabilidade de aceitar a nova solução como função da temperatura¹, de forma que a aceitação desse tipo de solução seja mais provável a altas temperaturas e bastante improvável a temperaturas reduzidas.

O sistema inicia em uma temperatura elevada, e a cada iteração ela é diminuída de forma que o equilíbrio térmico é atingido após uma determinada quantidade de iterações. A razão de decréscimo da temperatura é $T_n = a \cdot T_{n-1}$, observando-se empiricamente que o intervalo $0.8 \leq a \leq 0.99$ possibilita uma graduação adequada.

Finalmente, o algoritmo se encerra quando o sistema alcança estabilidade, condição na qual a temperatura está baixa e não é aceita mais nenhuma solução que piore o valor da melhor solução.

4.10.4. ALGORITMOS GENÉTICOS

Um algoritmo genético é descrito por Russel e Norvig (2003) como uma estratégia de busca local onde os estados sucessores são gerados pela combinação de dois estados pais. Os algoritmos genéticos foram desenvolvidos por John Henry Holland, que usou em sua concepção técnicas inspiradas pela biologia evolutiva, como hereditariedade, mutação, seleção natural e recombinação – aqui chamada de crossing over.

¹ Uma função da temperatura que pode ser empregada para o cálculo da probabilidade é o fator de Boltzmann, dada pela fórmula $e^{\Delta f / kT}$. Aqui, k é a constante de Boltzmann, que vale $1,3806503 \times 10^{-23}$ J/K.

De acordo com Holland (1975), um algoritmo genético inicia com um conjunto de k estados gerados aleatoriamente, que juntos formam uma população. Cada estado é um indivíduo dessa população, e é representado por uma cadeia sobre um alfabeto finito². Na etapa seguinte do algoritmo, cada indivíduo é avaliado por uma função de avaliação chamada função de fitness, que deve retornar valores mais altos para os melhores estados. Aplica-se, então, uma seleção aleatória de pares de indivíduos, de forma que a probabilidade de cada um deles ser selecionado é diretamente proporcional ao valor da função fitness.

Para ilustrar, vamos supor que nossa população inicial é composta de k indivíduos, identificados pelo conjunto $I = \{i_0, i_1, \dots, i_{k-1}, i_k\}$. Antes de aplicar a seleção dos pares, devemos avaliar cada um dos elementos do conjunto usando a função fitness $f(i)$, onde i representa um indivíduo. A partir desses resultados, determinaremos a probabilidade P_i de um elemento i ser escolhido usando a seguinte fórmula:

$$P_i = f(i) / (f(i_0) + f(i_1) + \dots + f(i_{k-1}) + f(i_k))$$

Para cada par selecionado, escolhe-se uma posição da cadeia ao acaso que dividirá cada indivíduo em duas cadeias que não possuirão necessariamente o mesmo número de elementos. Essas cadeias serão então recombinadas para formar dois novos indivíduos. A posição da cadeia selecionada é denominada ponto de crossover.

Continuando a ilustrar o processo, tomemos um par de indivíduos $\{i_1, i_2\}$ selecionados a partir do conjunto I de forma aleatória – respeitando, porém, suas respectivas probabilidades. Cada um deles possui 4 valores em sua cadeia e estão representados abaixo:

$$i_1 = c_1, c_2, c_3, c_4$$

$$i_2 = c_5, c_6, c_7, c_8$$

Assumindo que as posições dos elementos das cadeias são indexadas a partir de zero, vamos supor que a posição 2 foi escolhida ao acaso como ponto de crossover para os indivíduos i_1 e i_2 , respectivamente. Aplicando-se a recombinação das cadeias resultantes, teremos dois novos indivíduos i_3 e i_4 gerados com a seguinte configuração:

$$i_3 = c_1, c_2, c_7, c_8$$

$$i_4 = c_5, c_6, c_3, c_4$$

² Russel e Novig (2003) afirmam que é muito freqüente representar os indivíduos como uma cadeia de bits. Nesse caso, o alfabeto é composto pelos valores 0 e 1.

Ainda segundo Holland (1975), o algoritmo passa por uma etapa final, chamada de mutação, onde cada indivíduo da solução possui uma pequena probabilidade de ter um dos valores de sua cadeia permutados aleatoriamente. Como exemplo, uma mutação poderia ser aplicada selecionando-se ao acaso uma posição do indivíduo i , gerado anteriormente, e trocando-se o valor ocupado por essa posição por outro válido, porém diferente. Goldberg (1989) ressalta a importância da mutação para permitir que a população tenha maior variabilidade genética e impedir que a busca fique estagnada em um mínimo local.

4.11. TRABALHOS RELACIONADOS

A seguir são apresentados alguns trabalhos que se propõem a resolver algumas variações do Problema do Caixeiro Viajante. Mostrá-los aqui é importante para ilustrar as linhas de pesquisa que se tem realizado no tratamento de questões em comum e introduzir abordagens que não foram utilizadas neste projeto para atacar o problema, mas que são também perfeitamente aplicáveis.

4.11.1. "ANT COLONY SYSTEM: A COOPERATIVE LEARNING APPROACH TO THE TRAVELING SALESMAN PROBLEM" (Dorigo e Gambardella, 1997)

Este trabalho apresenta o algoritmo da otimização da colônia de formigas (ACS, do inglês Ant Colony System) e o aplica para resolver o PCV. Ele utiliza um conjunto de agentes inteligentes, denominados formigas, que cooperam entre si para encontrar soluções ótimas para o problema. Elas cooperam usando uma forma indireta de comunicação mediada por um feromônio que elas depositam nas arestas do grafo que modela o PCV enquanto constroem as soluções.

Os autores utilizaram a famosa biblioteca de instâncias do PCV denominada TSPLIB para aplicar o algoritmo e relatar as conclusões. O conjunto de resultados, mostrado na figura 3, faz um comparativo com outras heurísticas inspiradas na natureza, como o Simulated Annealing e os Algoritmos Genéticos, e os autores constatarem que o ACS oferece a melhor performance em quase todos os casos.

Problem name	ACS	GA	EP	SA	Optimum
Eil50 (50-city problem)	425 (427.96) [1,830]	428 (N/A) [25,000]	426 (427.86) [100,000]	443 (N/A) [68,512]	425 (N/A)
Eil75 (75-city problem)	535 (542.37) [3,480]	545 (N/A) [80,000]	542 (549.18) [325,000]	580 (N/A) [173,250]	535 (N/A)
KroA100 (100-city problem)	21,282 (21,285.44) [4,820]	21,761 (N/A) [103,000]	N/A (N/A) [N/A]	N/A (N/A) [N/A]	21,282 (N/A)

Figura 3: Comparativo entre o Ant Colony System (ACS), os Algoritmos Genéticos (GA), a Programação Evolucionária (EP) e o Simulated Annealing (AS) na resolução do Problema do Caixeiro Viajante. A primeira coluna da tabela exibe as instâncias do PCV que foram resolvidas. Os dados em negrito correspondem a melhor solução encontrada pelos algoritmos. Segundo esses dados, o ACS foi o que se mostrou mais eficiente.
Fonte: Dorigo e Gambardella, 1997.

4.11.2. "VARIOUS APPROACHES TO SOLVING THE TRAVELING SALESMAN PROBLEM" (Surana, 200-)

Neste trabalho, o autor utiliza três abordagens para a resolução do PCV, a saber: Abordagem de Força Bruta, que enumera todas as possíveis permutações das cidades e encontra o ciclo com menor caminho; Árvore de Extensão Mínima, que consiste em determinar o menor subgrafo que conecta todas as cidades do problema; e o Simulated Annealing, já descrito em seção anterior. Ao final, ele realiza uma comparação entre as abordagens em termos de qualidade da solução e tempo computacional. Suas conclusões são enumeradas a seguir:

- A abordagem da Árvore de Extensão mínima tem um custo significativamente baixo e provê boas soluções, até mesmo para um vasto número de cidades;
- A qualidade das soluções encontradas pelo Simulated Annealing diminui à medida em que a quantidade de cidades aumenta, na mesma proporção em que sua performance se deteriora;
- A qualidade das soluções encontradas pelo Simulated Annealing depende muito pouco da taxa de resfriamento do sistema. Da mesma forma, a temperatura inicial do sistema impacta muito pouco nesse quesito, a não ser

que esteja muito baixa no início da execução. Nesse caso, verifica-se influência negativa na qualidade das soluções.

4.11.3. "GENETIC ALGORITHM SOLUTION OF THE TRAVELING SALESMAN PROBLEM AVOIDING SPECIAL Crossover AND MUTATION" (Üçoluk, 1997)

O autor propõe, através deste trabalho, uma forma mais robusta e eficiente de realizar operações de crossover e mutação na utilização de Algoritmos Genéticos para a solução do PCV. Inicialmente, as formas padrão de realização dessas operações são apresentadas, juntamente com suas vantagens e desvantagens. Após isso, o método criado pelo autor é proposto e, finalmente, os resultados são apresentados, mostrando o ganho obtido com sua utilização no lugar dos métodos tradicionais. O estudo comparativo dos resultados mostra que o novo método supera o convencional por um fator de 2.2 em taxa de convergência.

5. METODOLOGIA

A metodologia começou a ser aplicada durante a concepção do projeto desta monografia. Inicialmente fez-se um levantamento bibliográfico com leitura, análise e triagem do material que compõe as referências, com o objetivo de dar suporte à definição do problema de pesquisa, à determinação dos objetivos e à construção do referencial teórico. A revisão contínua das obras pesquisadas deu apoio à execução das fases posteriores do trabalho. Durante a pesquisa, cada um dos algoritmos foi estudado com detalhes, bem como o problema em questão.

Em seguida, fez-se um levantamento das instâncias do PCV que foram estudadas. Esse levantamento foi realizado a partir da análise dos intervalos de valores das variáveis que compõem o problema, utilizando como base a definição apresentada no referencial teórico. As instâncias selecionadas, juntamente com os respectivos valores da solução ótima, estão listadas na tabela 1.

Uma vez definidas as instâncias, a etapa seguinte consistiu em uma pesquisa experimental laboratorial. Um computador foi utilizado para desenvolver um software utilizando a tecnologia Java que executa os algoritmos de busca selecionados para cada uma das diferentes instâncias do PCV tratada. A configuração do computador está descrita na tabela 2.

Com o software pronto, foi preciso realizar mais uma pesquisa bibliográfica por artigos e trabalhos relacionados com o tema de estudo, com o objetivo de precisar os valores dos parâmetros de execução dos algoritmos estudados comumente utilizados pela comunidade científica. Esses valores estão listados na tabela 3. Após isso, as instâncias foram executadas três vezes por algoritmo, com a ajuda do software desenvolvido.

Por fim, os dados de saída foram então analisados e organizados em forma de gráficos e tabelas, que ilustram o desempenho de cada um dos algoritmos quando executados para cada uma das instâncias. Esses dados foram utilizados como entrada para a construção das conclusões referentes ao trabalho, e serão apresentados na próxima seção.

TABELA 1 – Lista das 64 instâncias do PCV selecionadas.

Nome da instância	Solução ótima	Nome da instância	Solução ótima
a280	2579	pr1002	259045
berlin52	7542	pr107	44303
bier127	118282	pr124	59030
ch130	6110	pr136	96772
ch150	6528	pr144	58537
d1291	50801	pr152	73682
d1655	62128	pr226	80369
d198	15780	pr2392	378032
d2103	80450	pr264	49135
d493	35002	pr299	48191
d657	48912	pr439	107217
eil101	629	pr76	108159
eil51	426	rat195	2323
eil76	538	rat575	6773
fl1400	20127	rat783	8806
fl1577	22249	rat99	1211
fl417	11861	rd100	7910
gil262	2378	rd400	15281
kroA100	21282	rl1304	252948
kroA150	26524	rl1323	270199
kroA200	29368	rl1889	316536
kroB100	22141	st70	675
kroB150	26130	ts225	126643
kroB200	29437	tsp225	3916
kroC100	20749	u1060	224094
kroD100	21294	u1432	152970
kroE100	22068	u159	42080
lin105	14379	u1817	57201
lin318	42029	u2152	64253
linhp318	41345	u2319	234256
nrw1379	56638	u574	36905
p654	34643	u724	41910
pcb1173	56892	vm1084	239297
pcb442	50778	vm1748	336556

TABELA 2 – Configuração do computador utilizado para execução dos algoritmos.

Processador	Intel Core 2 Duo 2.80 GHz
Memória	2 GB
Sistema Operacional	Ubuntu 9.04
Java SDK	Versão 6, update 17

TABELA 3 – Parâmetros de execução dos algoritmos e seus valores

Simulated annealing	Temperatura inicial	10000
	Iterações após equilíbrio térmico (*)	100
Algoritmos genéticos	População inicial	50
	Porcentagem de mutação	5%
	Quantidade total de iterações	100

(*) O equilíbrio térmico é alcançado quando não há mudanças no valor da temperatura.

6. ANÁLISE DE RESULTADOS

As tabelas a seguir contêm os dados correspondentes ao resultado da execução dos dois algoritmos selecionados para estudo. As informações foram separadas conforme descrito abaixo:

- As tabelas 4 e 5 mostram os dados correspondentes à primeira execução do Simulated Annealing. As tabelas 6 e 7 mostram os dados correspondentes à segunda execução, e as tabelas 8 e 9, à terceira;
- As tabelas 10 e 11 mostram os dados correspondentes à primeira execução dos Algoritmos Genéticos. As tabelas 12 e 13 mostram os dados correspondentes à segunda execução, e as tabelas 14 e 15, à terceira.

É possível perceber que os resultados de cada execução foram divididos em duas tabelas, cada uma com 34 instâncias do PCV, em um total de 68 instâncias por execução. De acordo com o procedimento apresentado na metodologia, cada instância foi executada três vezes por algoritmo.

Os dados foram classificados em ordem crescente do nome da instância, mostrado na primeira coluna da tabela. As colunas seguintes representam, respectivamente, o tempo de execução de cada instância em milissegundos, o valor da solução encontrada e o erro relativo (Golub e Loan, 1996) à solução ótima, calculado de acordo com a fórmula abaixo:

$$E = (SE - SO) / SO$$

“E” é o valor do erro, “SE” é a solução encontrada e “SO” é a solução ótima. Por exemplo, a solução encontrada pela primeira execução da instância “a280” com o Simulated Annealing (primeira linha da tabela 4) foi 26549. De acordo com a tabela 1, o valor da solução ótima para essa instância é 2579. Usando a fórmula descrita acima, temos que o valor do erro com aproximação de três casas decimais é de 9,294. Se a solução encontrada fosse igual à solução ótima, o valor do erro seria um. Assim, quanto mais próximo de um estiver o erro, melhor a solução encontrada pelo algoritmo para a instância dada.

TABELA 4 – Simulated Annealing, 1ª execução (a280 a pcb442)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
a280	250	26549	9,294
berlin52	9	22424	1,973
bier127	1	639278	4,405
ch130	0	46030	6,534
ch150	1	55644	7,524
d1291	4	1740434	33,260
d1655	4182	1766161	27,428
d198	0	182494	10,565
d2103	5139	2608056	31,418
d493	401	360416	9,297
d657	790	662030	12,535
eil101	18	2710	3,308
eil51	3	1300	2,052
eil76	10	1884	2,502
fl1400	4	1706298	83,777
fl1577	6	1379276	60,993
fl417	1	494700	40,708
gil262	104	20708	7,708
kroA100	0	164979	6,752
kroA150	0	255535	8,634
kroA200	0	352321	10,997
kroB100	18	123706	4,587
kroB150	33	204381	6,822
kroB200	65	241697	7,211
kroC100	19	117697	4,672
kroD100	12	127602	4,992
kroE100	15	127912	4,796
lin105	0	128475	7,935
lin318	83	488720	10,628
linhp318	130	468204	10,324
nrv1379	3432	1105919	18,526
p654	599	1572357	44,387
pcb1173	3	1423233	24,016
pcb442	332	602212	10,860

TABELA 5 – Simulated Annealing, 1ª execução (pr1002 a vm1748)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
pr1002	1159	5153669	18,895
pr107	27	445897	9,065
pr124	33	517835	7,772
pr136	0	801544	7,283
pr144	0	831326	13,202
pr152	0	1025596	12,919
pr226	63	1295460	15,119
pr2392	5593	12733717	32,684
pr264	130	838703	16,069
pr299	114	572436	10,878
pr439	240	1524682	13,221
pr76	0	557623	4,156
rat195	0	22176	8,546
rat575	445	91832	12,559
rat783	2	175764	18,960
rat99	0	8379	5,919
rd100	13	46312	4,855
rd400	205	167662	9,972
rl1304	2003	7618708	29,120
rl1323	4	9699312	34,897
rl1889	3588	11766527	36,173
st70	0	3607	4,344
ts225	96	1239067	8,784
tsp225	56	32776	7,370
u1060	3	6681403	28,815
u1432	5	3976782	24,997
u159	0	468033	10,122
u1817	8	2151008	36,604
u2152	6876	2019392	30,429
u2319	13	5969346	24,482
u574	677	513852	12,924
u724	804	688734	15,434
vm1084	1594	6652660	26,801
vm1748	7	15036215	43,677

TABELA 6 – Simulated Annealing, 2ª execução (a280 a pcb442)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
a280	216	25249	8,790
berlin52	0	29929	2,968
bier127	23	529966	3,481
ch130	38	34178	4,594
ch150	20	44737	5,853
d1291	2061	1411053	26,776
d1655	7	2192291	34,287
d198	0	196027	11,422
d2103	10	3259160	39,512
d493	352	359707	9,277
d657	1	870085	16,789
eil101	16	2612	3,153
eil51	4	1212	1,845
eil76	0	2534	3,710
fl1400	4	1682953	82,617
fl1577	6	1339312	59,197
fl417	1	510367	42,029
gil262	0	26471	10,132
kroA100	16	137248	5,449
kroA150	25	191371	6,215
kroA200	73	244472	7,324
kroB100	0	170037	6,680
kroB150	35	188402	6,210
kroB200	0	342282	10,628
kroC100	11	131961	5,360
kroD100	0	157162	6,381
kroE100	15	128816	4,837
lin105	0	124307	7,645
lin318	0	553544	12,171
linhp318	157	458413	10,088
nrw1379	4	1432585	24,294
p654	1	2055470	58,333
pcb1173	1760	1129622	18,856
pcb442	276	624086	11,290

TABELA 7 – Simulated Annealing, 2ª execução (pr1002 a vm1748)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
pr1002	1195	5151679	18,887
pr107	0	545764	11,319
pr124	27	518754	7,788
pr136	32	650368	5,721
pr144	40	641009	9,950
pr152	0	1074888	13,588
pr226	73	1268684	14,786
pr2392	14	15225546	39,276
pr264	0	1155841	22,524
pr299	100	586922	11,179
pr439	1	1918610	16,895
pr76	0	585207	4,411
rat195	75	16810	6,236
rat575	440	83629	11,347
rat783	2	174578	18,825
rat99	0	7913	5,534
rd100	23	40342	4,100
rd400	266	166676	9,907
rl1304	4	9424754	36,260
rl1323	4	9958218	35,855
rl1889	4876	11670435	35,869
st70	0	3441	4,098
ts225	0	1595667	11,600
tsp225	80	33722	7,611
u1060	1877	5124645	21,868
u1432	5	3959572	24,885
u159	0	433631	9,305
u1817	8	2123084	36,116
u2152	3187	2161600	32,642
u2319	15	6054429	24,845
u574	1	646259	16,511
u724	739	707092	15,872
vm1084	1705	6767912	27,282
vm1748	4515	11971027	34,569

TABELA 8 – Simulated Annealing, 3ª execução (a280 a pcb442)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
a280	245	26247	9,177
berlin52	10	24882	2,299
bier127	33	517505	3,375
ch130	39	34069	4,576
ch150	0	53905	7,258
d1291	3091	1339425	25,366
d1655	5282	1715787	26,617
d198	89	154845	8,813
d2103	6235	2583595	31,114
d493	538	351343	9,038
d657	0	846947	16,316
eil101	0	3152	4,011
eil51	13	1241	1,913
eil76	18	1962	2,647
fl1400	6017	1231508	60,187
fl1577	4	1350229	59,687
fl417	1	510864	42,071
gil262	104	21386	7,993
kroA100	0	165298	6,767
kroA150	0	254574	8,598
kroA200	0	346501	10,799
kroB100	22	124201	4,610
kroB150	1	248133	8,496
kroB200	0	323783	9,999
kroC100	18	130045	5,268
kroD100	21	119683	4,621
kroE100	1	160457	6,271
lin105	0	115621	7,041
lin318	123	473812	10,273
linhp318	189	469727	10,361
nrv1379	3601	1104433	18,500
p654	1	2062571	58,538
pcb1173	2	1391075	23,451
pcb442	0	774344	14,250

TABELA 9 – Simulated Annealing, 3ª execução (pr1002 a vm1748)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
pr1002	1	6380624	23,631
pr107	0	560563	11,653
pr124	0	685236	10,608
pr136	0	806824	7,337
pr144	0	788178	12,465
pr152	0	949985	11,893
pr226	83	1298771	15,160
pr2392	8	15157075	39,095
pr264	0	1170780	22,828
pr299	154	579235	11,020
pr439	1	1928912	16,991
pr76	9	436942	3,040
rat195	0	20824	7,964
rat575	1	113776	15,798
rat783	1192	134538	14,278
rat99	0	7970	5,581
rd100	19	40133	4,074
rd400	297	162656	9,644
rl1304	3050	7384001	28,192
rl1323	2669	7698950	27,494
rl1889	5	14855655	45,932
st70	386	2714	3,021
ts225	1	1564494	11,354
tsp225	0	43896	10,209
u1060	2	6660093	28,720
u1432	3544	2990942	18,552
u159	0	460092	9,934
u1817	5506	1640594	27,681
u2152	7341	1983011	29,863
u2319	11814	4625248	18,744
u574	623	531475	13,401
u724	1	862572	19,582
vm1084	2	8506557	34,548
vm1748	4	14824865	43,049

TABELA 10 – Algoritmos Genéticos, 1ª execução (a280 a pcb442)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
a280	14015	32334	11,537
berlin52	1863	26041	2,453
bier127	5628	595788	4,037
ch130	5501	43013	6,040
ch150	6677	49816	6,631
d1291	80653	1740166	33,255
d1655	107606	2158458	33,742
d198	9246	177638	10,257
d2103	141343	3244783	39,333
d493	26666	434409	11,411
d657	36982	839200	16,157
eil101	4187	3120	3,960
eil51	1802	1546	2,629
eil76	2940	2250	3,182
fl1400	89884	1610394	79,012
fl1577	103976	1319058	58,286
fl417	22330	474111	38,972
gil262	12955	25440	9,698
kroA100	4115	158441	6,445
kroA150	6611	232057	7,749
kroA200	9382	332412	10,319
kroB100	4055	148432	5,704
kroB150	6647	236625	8,056
kroB200	9447	309183	9,503
kroC100	4038	157346	6,583
kroD100	4104	152945	6,183
kroE100	3995	155063	6,027
lin105	4249	110378	6,676
lin318	16000	559036	12,301
linhp318	16157	560420	12,555
nrv1379	88056	1413282	23,953
p654	37533	1955143	55,437
pcb1173	72753	1360052	22,906
pcb442	23511	746093	13,693

TABELA 11 – Algoritmos Genéticos, 1ª execução (pr1002 a vm1748)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
pr1002	60814	6350628	23,516
pr107	4423	497310	10,225
pr124	5232	627434	9,629
pr136	5833	746105	6,710
pr144	6330	761891	12,016
pr152	6825	970786	12,175
pr226	10869	1604062	18,959
pr2392	165138	15064803	38,851
pr264	12888	1069895	20,775
pr299	15029	736813	14,289
pr439	23687	1843507	16,194
pr76	2974	508398	3,700
rat195	9127	21742	8,359
rat575	31518	111910	15,523
rat783	45141	177602	19,168
rat99	4480	7864	5,494
rd100	4082	49817	5,298
rd400	21210	206898	12,540
rl1304	81180	9234908	35,509
rl1323	83567	9934723	35,768
rl1889	123981	14584512	45,075
st70	2661	3210	3,756
ts225	10858	1517975	10,986
tsp225	10790	39971	9,207
u1060	64984	6485360	27,940
u1432	92609	3913711	24,585
u159	7191	411843	8,787
u1817	118909	2058327	34,984
u2152	145448	2509431	38,055
u2319	154710	6037419	24,773
u574	32063	670496	17,168
u724	41455	844689	19,155
vm1084	65949	8435292	34,250
vm1748	115294	14721215	42,741

TABELA 12 – Algoritmos Genéticos, 2ª execução (a280 a pcb442)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
a280	13946	34793	12,491
berlin52	1850	25614	2,396
bier127	5518	602960	4,098
ch130	5530	41816	5,844
ch150	6637	56412	7,642
d1291	80114	1751461	33,477
d1655	105864	2156120	33,704
d198	9120	187096	10,857
d2103	138655	3288805	39,880
d493	26372	453035	11,943
d657	36979	847862	16,334
eil101	4082	3097	3,924
eil51	1798	1480	2,474
eil76	2972	2290	3,257
fl1400	87981	1685063	82,722
fl1577	100095	1330970	58,822
fl417	22017	508824	41,899
gil262	13016	26148	9,996
kroA100	4075	154699	6,269
kroA150	6636	263522	8,935
kroA200	9395	350707	10,942
kroB100	4008	150279	5,787
kroB150	6625	252635	8,668
kroB200	9293	325591	10,061
kroC100	3980	155896	6,513
kroD100	3999	159007	6,467
kroE100	4025	153470	5,954
lin105	4229	110612	6,693
lin318	15943	587515	12,979
linhp318	15957	576501	12,944
nrv1379	86156	1427321	24,201
p654	36974	2093930	59,443
pcb1173	71025	1398018	23,573
pcb442	23350	759760	13,962

TABELA 13 – Algoritmos Genéticos, 2ª execução (pr1002 a vm1748)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
pr1002	59974	6477738	24,006
pr107	4303	617907	12,947
pr124	5209	637591	9,801
pr136	5809	759674	6,850
pr144	6344	753317	11,869
pr152	6818	1057797	13,356
pr226	10862	1740073	20,651
pr2392	160194	15294069	39,457
pr264	12845	1137224	22,145
pr299	14970	777228	15,128
pr439	23138	1898632	16,708
pr76	2912	504809	3,667
rat195	8966	22848	8,836
rat575	31852	114854	15,958
rat783	44788	181958	19,663
rat99	4415	8910	6,358
rd100	4085	51999	5,574
rd400	20629	208374	12,636
rl1304	80200	9356611	35,990
rl1323	82775	10009089	36,043
rl1889	123002	14603370	45,135
st70	2686	3361	3,979
ts225	10755	1568210	11,383
tsp225	10798	40850	9,432
u1060	63276	6947191	30,001
u1432	91156	3972510	24,969
u159	7093	410249	8,749
u1817	115794	2102979	35,765
u2152	142151	2523412	38,273
u2319	153967	6023096	24,712
u574	32195	689090	17,672
u724	41273	865143	19,643
vm1084	65603	8482129	34,446
vm1748	111788	14905019	43,287

TABELA 14 – Algoritmos Genéticos, 3ª execução (a280 a pcb442)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
a280	13679	35633	12,817
berlin52	1838	26528	2,517
bier127	5536	601070	4,082
ch130	5449	43664	6,146
ch150	6504	59632	8,135
d1291	80098	1755534	33,557
d1655	107216	2151942	33,637
d198	9059	191795	11,154
d2103	138555	3286863	39,856
d493	26772	457344	12,066
d657	37020	850392	16,386
eil101	3999	3220	4,119
eil51	1763	1404	2,296
eil76	2942	2252	3,186
fl1400	87772	1699224	83,425
fl1577	100210	1338564	59,163
fl417	21731	511789	42,149
gil262	12963	26931	10,325
kroA100	3948	154510	6,260
kroA150	6617	276567	9,427
kroA200	9289	349704	10,908
kroB100	4028	154854	5,994
kroB150	6664	275198	9,532
kroB200	9312	330245	10,219
kroC100	4010	177670	7,563
kroD100	3972	168186	6,898
kroE100	3967	181370	7,219
lin105	4291	118969	7,274
lin318	15788	595597	13,171
linhp318	15978	593283	13,350
nrw1379	85219	1426381	24,184
p654	36854	2098460	59,574
pcb1173	72221	1400098	23,610
pcb442	23309	787988	14,518

TABELA 15 – Algoritmos Genéticos, 3ª execução (pr1002 a vm1748)

Nome da instância	Tempo de execução (ms)	Solução encontrada	Erro
pr1002	60502	6456928	23,926
pr107	4312	692560	14,632
pr124	5107	700550	10,868
pr136	5811	848861	7,772
pr144	6237	792052	12,531
pr152	6834	1103094	13,971
pr226	10816	1752543	20,806
pr2392	160322	15345937	39,594
pr264	12930	1173006	22,873
pr299	14869	793929	15,475
pr439	23089	1960174	17,282
pr76	2864	533045	3,928
rat195	8940	23560	9,142
rat575	31729	116032	16,132
rat783	44746	182257	19,697
rat99	4067	9144	6,551
rd100	3986	58354	6,377
rd400	20781	208345	12,634
rl1304	81736	9407913	36,193
rl1323	83551	9989156	35,970
rl1889	122507	14620841	45,190
st70	2663	3478	4,153
ts225	10747	1638055	11,934
tsp225	10779	43033	9,989
u1060	63312	6968043	30,094
u1432	90734	3972941	24,972
u159	7059	445431	9,585
u1817	115875	2107824	35,849
u2152	142023	2521592	38,245
u2319	156321	6022987	24,711
u574	31747	684482	17,547
u724	41300	875563	19,892
vm1084	67095	8492464	34,489
vm1748	112369	14878877	43,209

As tabelas apresentam os dados de forma bastante crua.⁸ Mas um fato que pode ser percebido é que em nenhuma das execuções o resultado do cálculo do erro foi um. Ou seja, a solução ótima não foi encontrada por nenhuma das implementações dos algoritmos construídas para este trabalho. No caso do Simulated Annealing, a fórmula utilizada para calcular o decréscimo da temperatura do sistema pode ter influenciado em sua eficiência. Os trabalhos encontrados durante a pesquisa relacionados ao tema utilizaram o mesmo cálculo e obtiveram eficiência semelhante. Uma nova busca por outros trabalhos poderia revelar outras fórmulas para o cálculo de decréscimo da temperatura do sistema que tornassem o algoritmo mais eficiente.

Já no caso dos Algoritmos Genéticos, a causa da baixa eficiência é conhecida e explicada a seguir. Nesse algoritmo, um cromossomo representa um caminho a ser percorrido pelo caixeiro no PCV. Ele é modelado como uma sequência de números situados entre um e a quantidade de cidades da instância do PCV resolvida. Por exemplo, a instância ilustrada na figura 1 contém quatro cidades, a saber: u, v, w e x. Vamos numerá-las como 1, 2, 3 e 4. O cromossomo utilizado para encontrar a solução para essa instância é modelado com cinco dígitos, onde cada dígito é um número situado entre um e quatro. Uma das combinações possíveis é mostrada abaixo:

1, 3, 2, 4

Cada indivíduo da população inicial gerada antes da execução do algoritmo contém um cromossomo. O problema é que o processo de geração aleatória de indivíduos desenvolvido neste trabalho não leva em conta a criação de cromossomos que representam caminhos equivalentes. Como exemplo, o caminho 3, 2, 4, 1 é uma simples rotação do ilustrado acima e, portanto, equivalente a aquele. A conclusão é que se o algoritmo utilizado na geração aleatória de indivíduos pudesse descartar os semelhantes conforme descrito, a eficiência do algoritmo certamente aumentaria. Porém, o impacto negativo no tempo de execução, que no caso dos Algoritmos Genéticos já foi alto em relação ao Simulated Annealing, não poderia ser desprezado.

Alguns gráficos foram gerados para ilustrar melhor as conclusões deduzidas a partir da análise dos resultados. A figura 4 mostra um comparativo entre o tempo médio de execução de cada algoritmo. O eixo x contém a quantidade de cidades de cada instância em ordem crescente, e o eixo y mostra o tempo médio, calculado a partir da média aritmética simples entre os três tempos de execução dos algoritmos. Por exemplo, para a primeira instância, que contém 51 cidades, os tempos de execução do Simulated Annealing foram 3, 4, e 13

milissegundos. O tempo médio foi, portanto, aproximadamente 6,667. De acordo com o gráfico, o tempo de execução do Simulated Annealing se mantém aproximadamente constante conforme cresce o número de cidades, tendo um modesto aumento a partir de mil cidades. O tempo médio do Algoritmo Genético, por outro lado, cresce de forma exponencial em relação ao número de cidades.

A figura 5 ilustra o valor médio das soluções encontradas em função do número de cidades. Esse valor é calculado de forma semelhante ao tempo médio de execução, utilizando-se os valores das soluções encontradas no lugar dos tempos. A linha preta é composta pelas soluções ótimas definidas para cada instância, conforme a tabela 1. A linha azul exibe os valores médios das soluções encontradas pelo Simulated Annealing, enquanto a linha vermelha exibe os valores médios encontrados pelo Algoritmo Genético. Observa-se que ambos tiveram eficiência semelhante, com o Simulated Annealing apresentando resultados ligeiramente melhores. A eficiência de ambos, porém, piora de forma não linear, encontrando soluções mais distantes da solução ótima conforme cresce o número de cidades da instância do PCV resolvida.

A figura 6 é um gráfico que exibe um comparativo entre o erro médio em relação à solução ótima dos dois algoritmos, calculado como uma média aritmética dos erros encontrados para as três execuções de cada um deles. A linha azul corresponde ao erro médio do Simulated Annealing, e a linha verde, do Algoritmo Genético. Pode-se observar que o erro de ambos cresce proporcionalmente ao número de instâncias, embora os erros calculados para o Simulated Annealing sejam ligeiramente menores.

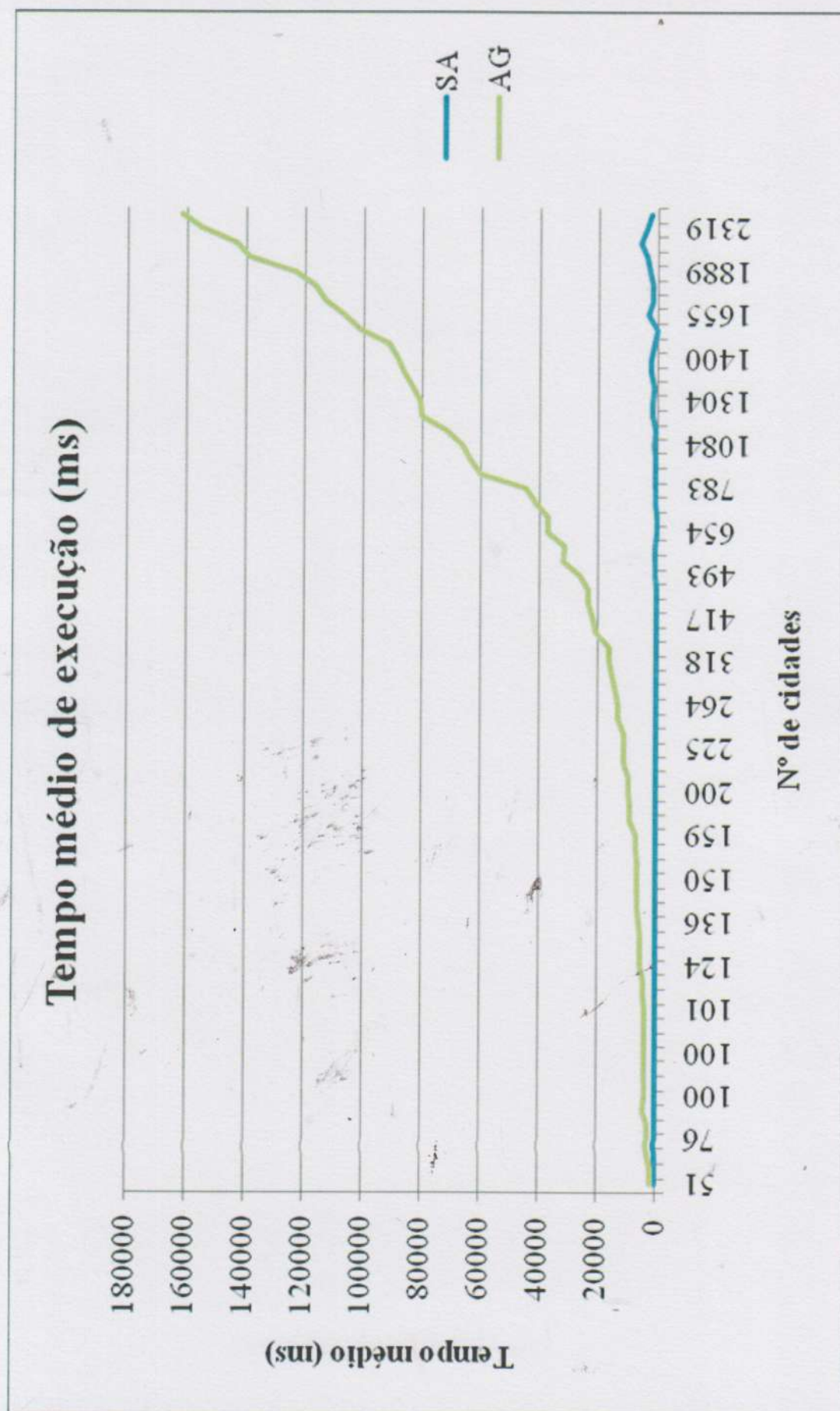


Figura 4 – Tempo médio de execução em função da quantidade de cidades do PCV.

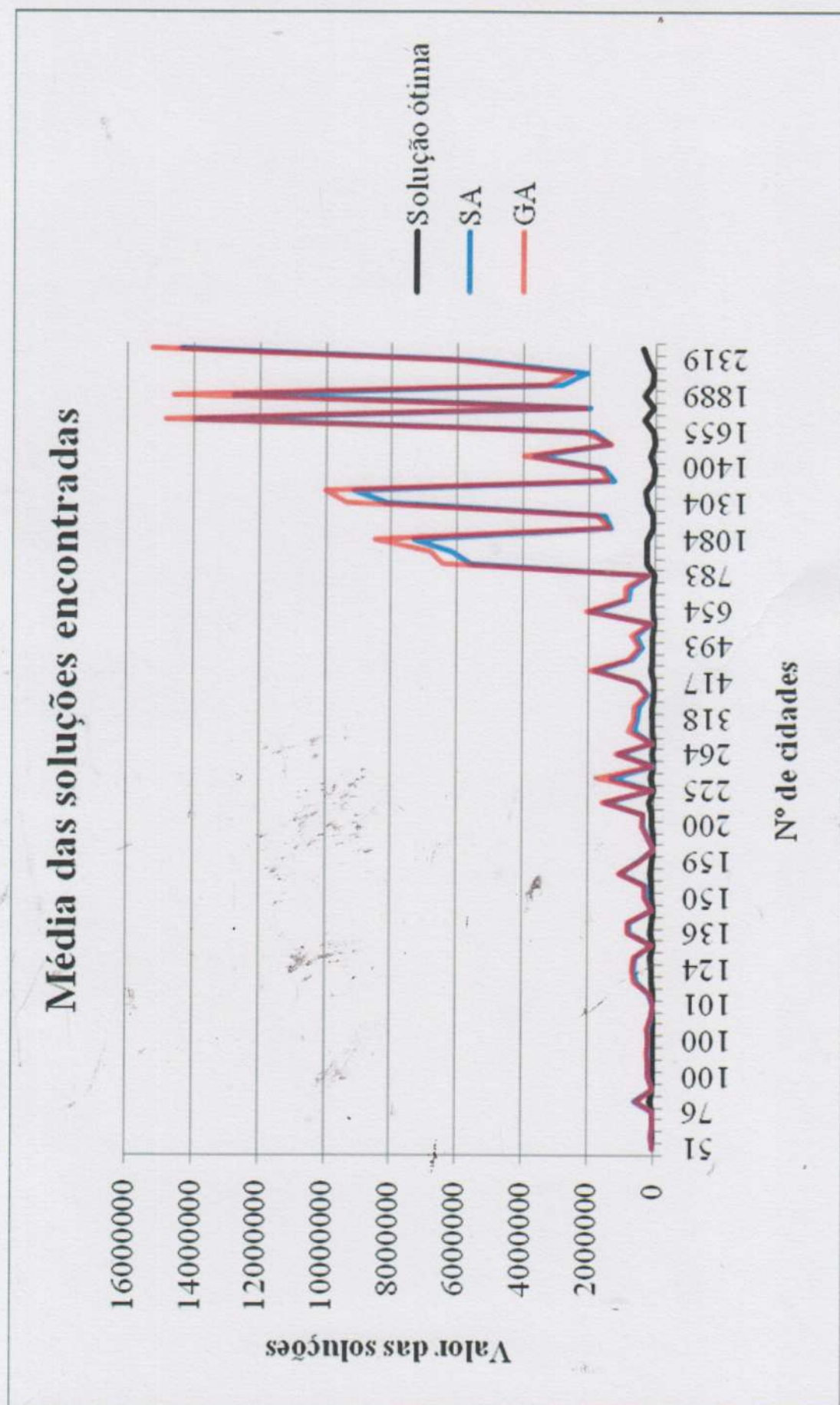


Figura 5 – Valores médios das soluções em função da quantidade de cidades do PCV.

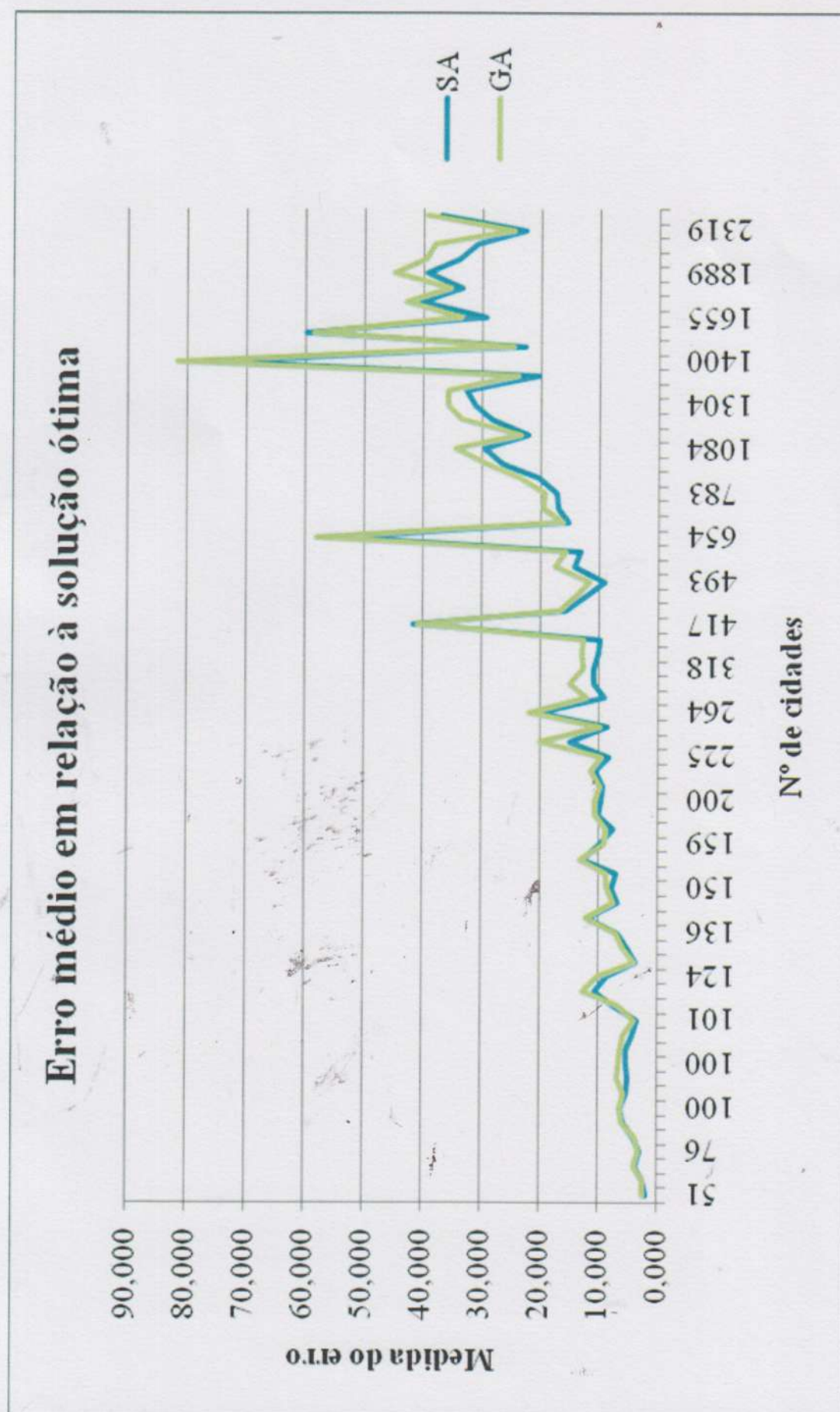


Figura 6 – Erro médio relativo à solução ótima em função da quantidade de cidades do PCV.

7. CONCLUSÃO E PONTOS DE PARTIDA

Uma vez realizado o trabalho de coleta dos dados resultantes da execução dos algoritmos – tendo estes como entradas as instâncias do PCV selecionadas e apontadas – e finalizada a análise crítica desses dados, o pesquisador mune-se dos fatos enumerados na seção anterior para chegar às conclusões expostas a seguir.

A experiência de implementação dos algoritmos para este trabalho mostra que os Algoritmos Genéticos são, em termos de esforço de desenvolvimento, mais complexos do que o Simulated Annealing, pois envolvem sub-operações – como mutação, reprodução e geração aleatória de soluções iniciais – que não são triviais. Já este último requer menos esforço de desenvolvimento porque a heurística envolvida é mais intuitiva e direta.

Quanto menos oneroso é um algoritmo na realização de sua tarefa, ou seja, quanto menos recursos computacionais ele consome durante sua execução, maior é sua medida de desempenho. Isso exposto, os resultados mostram que o desempenho do Simulated Annealing em relação aos Algoritmos Genéticos fica cada vez maior conforme cresce a quantidade de cidades contidas na instância do PCV a ser resolvida, já que seu tempo de execução se mantém praticamente constante. Esse fato é consequência direta da diferença de complexidade de implementação entre os mesmos, mencionada no parágrafo anterior. Os Algoritmos Genéticos precisam de uma população inicial muito grande para alcançar uma eficiência satisfatória, e isso acarreta em alto consumo de recursos computacionais, como memória e tempo de processamento.

No que tange à eficiência, quanto mais próxima da solução ótima é a solução encontrada por um algoritmo, mais eficiente ele se torna. É possível concluir dos resultados apresentados que tanto o Simulated Annealing quanto os Algoritmos Genéticos são equivalentemente eficientes na resolução do PCV, porém o primeiro carrega consigo uma leve vantagem nesse ponto em relação ao segundo. Esse fato é observado principalmente na resolução de instâncias com quantidades maiores de cidades. Não é desprezível, no entanto, o impacto positivo na eficiência de ambos que poderia acontecer caso as otimizações apresentadas na seção anterior fossem aplicadas. Mas os ganhos seriam equivalentes tanto para um quanto para outro.

Respeitando-se o contexto sob o qual essa pesquisa foi elaborada, esses fatos permitem concluir, portanto, que o Simulated Annealing é uma melhor escolha de busca heurística para resolver o PCV em comparação com os Algoritmos Genéticos.

7.1. MELHORIAS E PONTOS DE PARTIDA

Esta seção tem o objetivo de expor tanto pontos que podem ser melhorados nesta pesquisa quanto tópicos a partir dos quais novos trabalhos podem ser derivados.

Como já foi mostrado, os resultados gerados pelas implementações dos algoritmos construídas pelo pesquisador encontraram soluções para as instâncias do PCV relativamente distantes das soluções ótimas correspondentes. O leitor interessado pode substituir a implementação dos algoritmos por outra qualquer, ou mesmo obter a que já existe e incrementá-la, explorando as sugestões apresentadas na análise de resultados.

Há dezenas de algoritmos de otimização disponíveis para resolver problemas que se encaixam na categoria à qual o PCV pertence. É possível continuar o trabalho acrescentando estudos de desempenho relativos a esses algoritmos. Como exemplo temos a Otimização da Colônia de Formigas e a Busca Tabu.

O Problema de Roteamento de Veículos (PRV) engloba dois problemas clássicos da computação: o Problema do Caixeiro Viajante, estudo deste trabalho, e o Bin Packing Problem, onde é preciso distribuir um monte de lixo entre a menor quantidade possível de lixeiras. No PRV é dada uma frota de veículos com capacidade C , uma carga Q e um conjunto de N cidades para as quais os veículos devem distribuir essa carga, partindo de um depósito inicial D e retornando a este após a realização da entrega. Deve ser utilizada a menor quantidade de veículos possível, e a rota a ser percorrida por eles deve ter a menor distância dentre todas as rotas enumeráveis. Um trabalho que poderia ser realizado a partir deste é o estudo de desempenho de diversos algoritmos de busca na resolução do PRV e suas variantes.

Uma sugestão final, mas não menos interessante, diz respeito às variantes assimétricas das instâncias do PCV introduzidas no referencial teórico. Os algoritmos de estudo desse trabalho e os sugeridos no parágrafo anterior podem ser perfeitamente adaptados para resolver essas instâncias, e os resultados, analisados e comparados nos mesmos moldes.

NEWELL, A.; SIMON, H. A. The search for generality. In: IFIP CONGRESS, 1, 1965, Chicago. Proceedings... Chicago: Spartan Books, 1965, p. 17-24.

RUSSEL, Stuart; NORVIG, Peter. Inteligência Artificial, 2. ed. Rio de Janeiro: Campus, 2003.

SURANA, Sahil. Various Approaches to Solving Traveling Salesman Problem. [200-]. Dissertação - [S.B].

8. BIBLIOGRAFIA

- CORMEN, Thomas H. *et al.* **Algoritmos: teoria e prática**. 2. ed. Rio de Janeiro: Elsevier, 2002.
- DECHTER, R.; PEARL, J. Generalized best-first search strategy and the optimality of A*. **Journal of the Association for Computing Machinery**, [S.l], v.3, n.32, p.506-536, 1985.
- DORIGO, Marco; GAMBARDELLA, Luca Maria. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. **IEEE Transactions on Evolutionary Computation**, [S.l], v.1, n.1, p.53-66, 1997.
- GOLDBERG, David E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. EUA: Addison-Wesley, 1989.
- GOLUB, Gene H.; LOAN, Charles V. Van. **Matrix Computations**. 3. ed. Baltimore: The Johns Hopkins University Press, 1996.
- HART, P. E.; NILSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE Transactions on Systems Science and Cybernetics**, [S.l], v.4, n.2, p.100-107, 1968.
- _____. Correction to "A formal basis for the heuristic determination of minimum cost paths". **SIGART Newsletter**, [S.l], n.37, p.28-29, 1972.
- HOLLAND, J. H. **Adaption in Natural and Artificial Systems**. Michigan: University of Michigan Press, 1975.
- KIRKPATRICK, S.; GELLAT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science**, [S.l], n.220, p.671-680, 1983.
- LIN, S. Computer solutions of the traveling salesman problem. **Bell Systems Technical Journal**, [S.l], v.44, n.10, p.2245-2269, 1965.
- MOORE, E. F. The shortest path through a maze. In: INTERNATIONAL SYMPOSIUM ON THE THEORY OF SWITCHING, 2., 1959, Massachusetts. **Proceedings...** Massachusetts: Harvard University Press, 1959, p. 285-292.
- NEWELL, A.; EMST, G. The search for generality. In: IFIP CONGRESS, 1., 1965, Chicago. **Proceedings...** Chicago: Spartan Books, 1965, p. 17-24.
- RUSSEL, Stuart; NORVIG, Peter. **Inteligência Artificial**. 2. ed. Rio de Janeiro: Campus, 2003.
- SURANA, Sahil. **Various Approaches to Solving Traveling Salesman Problem**. [200-]. Dissertação – [S.l].

ÜÇOLUK, Göktürk. **Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation.** 1997. Dissertação – Ankara, Turquia.