UNIVERSIDADE PRESBITERIANA MACKENZIE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Ricardo Fábio Sato

PLASTICIDADE SINÁPTICA E O PROBLEMA DO CAIXEIRO VIAJANTE

Orientador: Prof. Dr. Luiz Henrique Alves Monteiro

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Presbiteriana Mackenzie como requisito para obtenção do título de Mestre em Engenharia Elétrica

São Paulo
Junho – 2010

S253p Sato, Ricardo Fábio.
Plasticidade sináptica e o problema do caixeiro viajante / Ricardo Fábio Sato – 2010.
88 f. : il. ; 30 cm

Dissertação (Mestrado em Engenharia Elétrica) – Universidade Presbiteriana Mackenzie, São Paulo, 2010.
Bibliografia: f. 65-67.

1. Plasticidade sináptica. 2. Problema do caixeiro viajante. 3. Rede neural. 4. Regra de Hebb. I. Título.

CDD 006.32

Resumo

Propõem-se algoritmos para solucionar o clássico problema de otimização chamado de o problema do caixeiro viajante. Esses algoritmos são inspirados no fato de que sinapses que conectam neurônios biológicos podem se alterar com o passar do tempo em função da atividade desses neurônios. Assim, fazem-se analogias entre neurônios e cidades a serem visitadas pelo caixeiro, e entre estradas e sinapses que conectam os neurônios. Os desempenhos dos algoritmos propostos são avaliados comparando-se os comprimentos dos caminhos gerados por eles com o comprimento do caminho ótimo. Os resultados obtidos em testes realizados em instâncias geradas aleatoriamente e em dez instâncias de base de dados pública indicam que essa é uma abordagem promissora.

Palavras-chave: *plasticidade sináptica, problema do caixeiro viajante, rede neural, regra de Hebb.*

Abstract

Algorithms for solving the classical optimization problem called traveling salesman problem are proposed. These algorithms are inspired on the fact that synapses connecting biological neurons can be altered as the time passes in function of the neuronal activity. Thus, analogies between neurons and cities to be visited by the salesman and between roads and synapses coupling neurons are made. The performances of the proposed algorithms are evaluated by comparing the lengths of the paths produced by them with the length of the optimal solution. The results obtained in experiments performed by using randomly-generated instances and ten instances from a public database suggest that this is a promising approach.

Keywords: *Hebb's rule, neural network, synaptic plasticity, traveling salesman problem*.

SUMÁRIO

1 INTRODUÇÃO 5

2 CONCEITOS PRELIMINARES 7

2.1 MODELOS BÁSICOS DE NEURÔNIO 7

2.2 A REGRA DE HEBB 10

2.3 O PROBLEMA DO CAIXEIRO VIAJANTE 11

3 ALGORITMOS PROPOSTOS 13

3.1 A REDE NEURAL ARTIFICAL 13

3.2 AS REGRAS DE MODIFICAÇÃO SINÁPTICA 14

3.3 OS ALGORITMOS PROPOSTOS 16

4 EXPERIMENTOS NUMÉRICOS 25

4.1 INTRODUÇÃO 25

4.2 INSTÂNCIAS GERADAS ALEATORIAMENTE 25

4.3 INSTÂNCIAS DA TSPLIB 31

5 CONCLUSÕES 64

BIBLIOGRAFIA 65

APÊNDICE A 68

APÊNDICE B 82

1 INTRODUÇÃO

O cérebro humano é composto de aproximadamente 10^{11} neurônios ligados por 10^{15} sinapses. Apenas pela sua dimensão, uma rede dessa natureza já representa um obstáculo para seu estudo (SPORNS e ZWI, 2004). Além disso, a complexidade dos processos químicos e elétricos envolvidos na interação entre os neurônios colabora para aumentar a dificuldade de se compreender o funcionamento do cérebro.

Sabe-se que o funcionamento (normal ou patológico) do sistema nervoso depende fortemente das conexões sinápticas que existem entre os neurônios. Assim, tanto a topologia quanto a intensidade dessas conexões afetam o desempenho de funções típicas desse sistema, como aprendizagem e memória. Há evidências experimentais que as características das conexões neuronais são modificadas em função dos estímulos recebidos (BUONOMANO e MERZENICH, 1998). Por exemplo, Bear (2003) relata que quando um neurônio do córtex visual recebe informação correlacionada dos dois olhos, ele se torna sensível a estímulos captados por ambos os olhos. Quando essa correlação não existe, como no caso de estrabismo, então esse neurônio só responde a estímulos vindos de um dos olhos. Outro exemplo clássico, descoberto por volta de 1970, refere-se à potenciação de longa duração (*long-term potentiation* – LTP), pela qual repetidos estímulos elétricos periódicos (com frequência da ordem de 10-100 Hz) de curta duração (da ordem de 1-10 segundos) produzem um aumento de longa duração (da ordem de 1-10 horas) na intensidade da sinapse estimulada, localizada, nos primeiros experimentos, no hipocampo de coelhos (BEAR et al., 2002).

Considerado pioneiro em estudos sobre plasticidade sináptica, D. O. Hebb, em 1949, propôs uma regra de intensificação de sinapses excitatórias, afirmando que se um neurônio (pré-sináptico) colabora para o disparo de outro (pós-sináptico), então a sinapse que os conecta tende a se fortalecer. Contudo, sinapses excitatórias que só se fortalecem não podem por si só explicar a dinâmica constatada na atividade neural (GERSTNER e KISTLER, 2002). Há inúmeras evidências que elas, na verdade, se enfraquecem quando a correlação entre as atividades dos neurônios pré e pós-sináptico é “pequena”, o que é conhecido como depressão de longa duração (*long-term depression* – LTD) (SONG et al., 2000). Sinapses inibitórias se alteram de maneira similar: elas se intensificam quando há disparo síncrono dos neurônios pré e pós-sináptico e se enfraquecem caso contrário (SIRI et al., 2007). Em conjunto, essas regras sobre modificação sináptica constituem uma base importante para a teoria de aprendizado

não-supervisionado de sistemas neurais (FIORI, 2005) sendo, portanto, parte fundamental deste trabalho.

Aqui, utilizamos regras locais de modificação sináptica para solucionar o problema do caixeiro viajante (LIN e KERNIGHAN 1973; SCHRIJVER, 2005), fazendo-se um paralelo entre cidade e neurônio; peso sináptico entre dois neurônios e custo para se viajar entre duas cidades.

Este trabalho está assim estruturado. No capítulo 2, trata-se de alguns conceitos básicos, como modelos de neurônio e regra de Hebb, e introduz-se o problema do caixeiro viajante. No capítulo 3, descrevem-se o modelo e os algoritmos desenvolvidos, baseados em neuroplasticidade, para solucionar o problema do caixeiro viajante. No capítulo 4, apresentam-se os experimentos realizados e os resultados obtidos por simulação numérica. E, finalmente, no capítulo 5 apontam-se as conclusões.

2 CONCEITOS PRELIMINARES

2.1 MODELOS BÁSICOS DE NEURÔNIO

O cérebro realiza uma computação complexa, não-linear e paralela, permitindo que ele possa realizar tarefas como reconhecimento de padrões e controle motor, de forma rápida e eficaz (HAYKIN, 1998). Por conta dessas qualidades, o cérebro serviu de inspiração para a criação de redes neurais artificiais. Elas podem ser definidas como um sistema computacional desenvolvido para modelar o funcionamento do cérebro para a realização de uma determinada tarefa. A concepção moderna de redes neurais teve início através dos trabalhos de W. McCulloch e W. Pitts, em 1943. McCulloch era um psiquiatra especialista em anatomia neural e Pitts, um matemático. Juntos, eles criaram um modelo básico de neurônio artificial que serviu de inspiração para a construção de redes neurais artificiais (HAYKIN, 1998).

Simplificadamente, o cérebro é composto por um conjunto de neurônios, interligados através de sinapses, formando uma rede complexa. As informações que trafegam por essa rede são formadas, biologicamente, por sinais de naturezas elétrica e química. Dentro desse contexto, um neurônio constitui a menor unidade de processamento de informação. Já a sinapse representa a ligação entre dois neurônios, o que possibilita o fluxo das informações pela rede. Uma propriedade importante das sinapses é sua capacidade adaptativa, pois se pode modificar sua intensidade (seu peso sináptico) conforme as circunstâncias. Essa propriedade recebe o nome de plasticidade sináptica.

Assim, a plasticidade sináptica de uma rede neural relaciona-se com a variação temporal dos pesos das sinapses. Tais pesos ponderam os sinais de entrada dos neurônios da rede, dado que a entrada de um neurônio é formada pela soma de cada estímulo recebido de outro neurônio multiplicado pelo peso correspondente da sinapse que os interliga. Um peso positivo corresponde à sinapse excitatória; um peso negativo à sinapse inibitória.

Da mesma forma, os estímulos recebidos por um neurônio artificial, multiplicados pelos respectivos pesos, são somados, o que forma o sinal de entrada a ser processado pelo neurônio. Assim como o neurônio biológico, o artificial possui um mecanismo que limita a amplitude do seu sinal de saída. O valor de entrada que provoca a mudança no comportamento do neurônio, do estado de repouso para o estado ativo, recebe o nome de valor limiar ou simplesmente limiar. Em termos matemáticos, esse processamento é definido

por uma função. Dado que essa função determina se o neurônio é ativado ou não, ela recebe o nome de função de ativação. As principais funções de ativação utilizadas nas redes neurais artificiais são: função limiar, função linear por partes com saturação e função sigmoidal (HAYKIN, 1998).

Para um neurônio *k*, a função de ativação limiar, também chamada de função degrau, representa a transição abrupta entre dois níveis (usualmente, 0 e 1) na saída y_k quando a entrada v_k excede o limiar (usualmente o limiar vale 0). Assim:

$$y_k = \begin{cases} 1 \text{ se } v_k \geq 0 \\ 0 \text{ se } v_k < 0 \end{cases} \qquad (2.1)$$

A figura 2.1 apresenta o gráfico da função de ativação limiar dada pela equação 2.1, sendo que *φ(v)* indica tal função e *v* a entrada do neurônio:

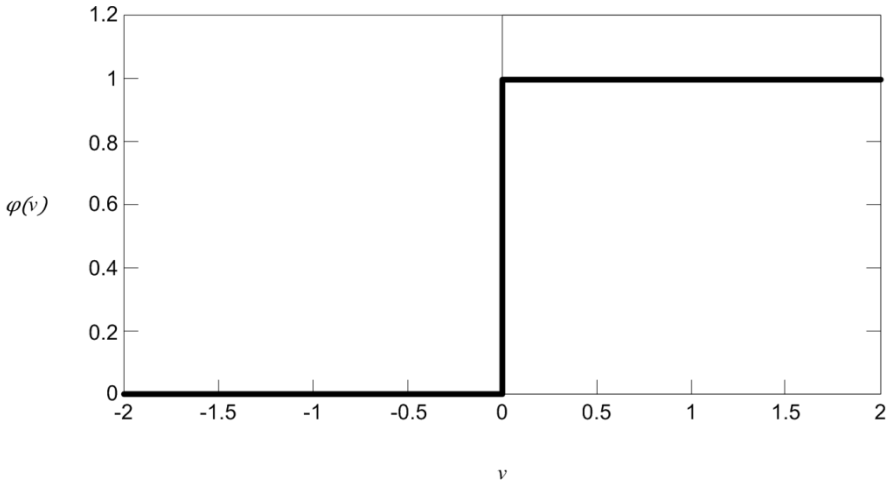

Figura 2.1: Função de ativação limiar. A saída do neurônio está representada no eixo vertical *φ(v)* e a entrada do neurônio está representada no eixo horizontal *v* (HAYKIN, 1998).

A função linear por partes com saturação apresenta forte semelhança com a função limiar; contudo, a transição entre os dois níveis é feita de forma contínua e linear, de acordo com a expressão:

$$y_k = \begin{cases} 1 \text{ se } v_k \geq +\frac{1}{2} \\ v_k \text{ se } +\frac{1}{2} > v_k > -\frac{1}{2} \\ 0 \text{ se } v_k \leq -\frac{1}{2} \end{cases} \qquad (2.2)$$

A figura 2.2 representa o gráfico dessa função de ativação linear por partes.

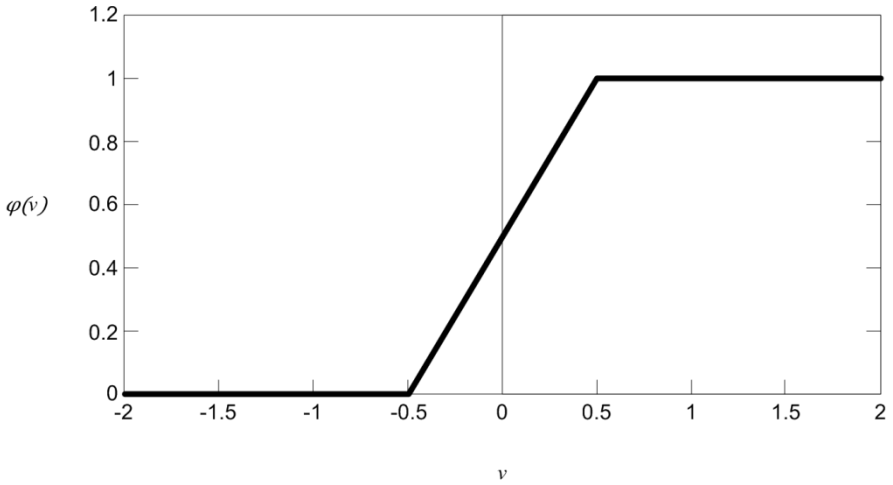

Figura 2.2: Função de ativação linear por partes com saturação. A saída do neurônio está representada no eixo vertical *φ(v)* e a entrada do neurônio está representada no eixo horizontal *v* (HAYKIN, 1998).

A função sigmoidal apresenta uma transição suave entre níveis (de modo que a função, além de contínua, é diferenciável), com a forma da letra S, dada por:

$$y_k = \frac{1}{1 + \exp(-a\, v_k)} \qquad (2.3)$$

A figura 2.3 ilustra a função de ativação sigmoidal para *a=1*.

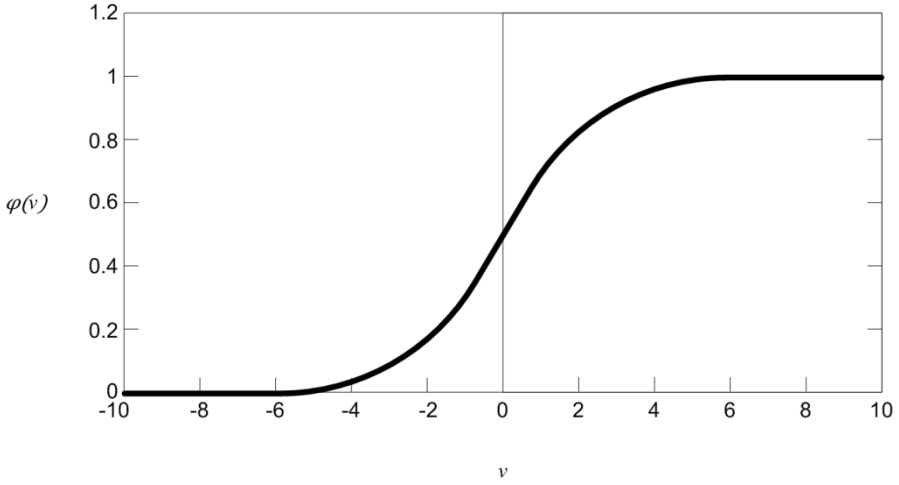

Figura 2.3: Função de ativação sigmoidal. A saída do neurônio está representada no eixo vertical *φ(v)* e a entrada do neurônio está representada no eixo horizontal *v* (HAYKIN, 1998).

A próxima seção trata de como neurônios podem “interagir”, quando estão ligados em rede, apresentando o mecanismo utilizado que se acredita ser a base do aprendizado.

2.2 A REGRA DE HEBB

Hebb trabalhou na área da neuropsicologia, descrevendo de forma pioneira o mecanismo de modificação sináptica. Ele escreveu sua idéia no livro “*The Organization of Behavior*”, publicado originalmente em 1949, da seguinte forma:

> When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased. (GERSTNER e KISTLER, 2002).

Portanto, segundo Hebb, se um neurônio provoca (colabora para) o disparo de outro, então há o fortalecimento da sinapse entre ambos. Essa proposição apóia-se no conceito de sincronismo (ou talvez seja mais preciso dizer, na correlação) entre disparos neurais. Com tal teoria, Hebb tentou explicar a relação entre atividade neuronal e o processo de aprendizagem de comportamentos. Apesar de a teoria ter sido formulada sem o respaldo de comprovação prática, ela foi se consolidando na medida em que descobertas experimentais foram ocorrendo. Diante disso, essa proposta alcançou tamanha importância que ficou conhecida como regra de Hebb (GERSTNER e KISTLER, 2002).

O estudo de Hebb tratava de sinapses excitatórias, mas sinapses inibitórias parecem se comportar de modo semelhante; ou seja, elas se intensificam se há o disparo síncrono entre os neurônios pré e pós-sináptico e se enfraquecem caso contrário (SIRI et al., 2007).

Como a regra de Hebb trata do fortalecimento e do enfraquecimento de sinapses, em função das atividades dos neurônios, a plasticidade sináptica tornou-se a base no estudo da maioria dos modelos de aprendizado e memória de redes neurais. Como as sinapses ligam os neurônios, a plasticidade sináptica desempenha um papel poderoso, pois ela permite modificar as propriedades das respostas da rede neural (ABBOTT e NELSON, 2000).

Várias simulações computacionais e estudos analíticos têm sido realizados com o objetivo de compreender como a plasticidade sináptica afeta a evolução temporal e a topologia de uma rede neural. Por exemplo, Chao e Chen (2005) usaram neurônios cuja saída

é uma função linear por partes da entrada a fim de estudar numericamente como modificações sinápticas baseadas na regra de Hebb influenciam a frequência de sincronismo de uma rede formada por 40 a 200 neurônios sujeitos a uma entrada constante. Sporns (2006) propôs um modo de gerar padrões fractais de conexão neuronal em redes com 128 neurônios, e comparou seus resultados com dados da literatura. Kempter et al. (1999), Roberts (1999) e Gerstner e Kistler (2002) estudaram, analiticamente, como a correlação entre os disparos dos neurônios pré e pós-sináptico modifica tal sinapse, usando diferentes modelos de neurônios. Siri et al. (2007) simularam o efeito da regra de Hebb numa rede com 500 neurônios conectados aleatória e esparsamente, por sinapses excitatórias e inibitórias. Morrison et al. (2008) apresentaram e analisaram diversos modelos de plasticidade sináptica baseados em equações diferenciais.

A seguir, descreve-se um clássico problema de otimização: o problema do caixeiro viajante.

2.3 O PROBLEMA DO CAIXEIRO VIAJANTE

O problema do caixeiro viajante é um exemplo clássico de problema de otimização combinatória, pertencente à classe NP-completo, que vem sendo estudado há pelo menos 150 anos (LIN e KERNIGHAN, 1973). Sua formulação pode ser descrita conforme a seguir. Considere N cidades e o custo de ir de uma cidade para outra. O custo pode ser, por exemplo, a distância, o gasto com combustível ou o tempo de viagem. O objetivo é determinar o percurso fechado que visita cada cidade uma única vez com custo mínimo e que retorna à cidade de partida. Se o custo considerado de ir da cidade c_1 para a cidade c_2 é o mesmo, tem-se a formulação do problema do caixeiro viajante simétrico. Por outro lado, se o custo de ir da cidade c_1 para a cidade c_2 é diferente do custo de se ir de c_2 para c_1, tem-se o caso assimétrico (ONCANA et al., 2009). Nesse trabalho, trata-se do caso simétrico, no qual esses custos são iguais.

Considerando N o número de cidades a se visitar, determinar a solução ótima através de busca exaustiva é uma tarefa computacionalmente inviável se N é grande (nesse contexto, "grande" significa da ordem de 20 ou maior), dado que o número de caminhos possíveis vale $(N-1)!$. Por exemplo, para $N=20$, há mais de $1{,}21\times10^{17}$ percursos possíveis e encontrá-los consumiria um tempo de processamento da ordem de meses num computador pessoal típico dos dias de hoje.

Existem inúmeros algoritmos que procuram resolver, ao menos aproximadamente (BAGCHI et al., 2006), esse problema, como redes neurais artificiais de Hopfield (HAYKIN, 1998), algoritmos baseados em colônias de formigas (DORIGO et al., 1996) e algoritmos genéticos (POTVIN, 1996). Normalmente, a solução encontrada é aproximada (sub-ótima).

Aqui, propõem-se algoritmos para solucionar o problema do caixeiro viajante simétrico, baseados em regras de modificação sináptica encontradas em redes neurais biológicas. Assume-se que cada cidade corresponde a um neurônio da rede e a intensidade da sinapse que conecta dois neurônios é tomada como o inverso da distância de uma cidade para outra. Assim, sinapses com intensidades maiores correspondem aos caminhos com distâncias mais curtas. Esses algoritmos são apresentados no próximo capítulo.

3 ALGORITMOS PROPOSTOS

3.1 A REDE NEURAL ARTIFICAL

Neste trabalho, busca-se fazer um paralelo entre cidades conectadas por uma malha viária, com os respectivos custos associados às viagens entre essas cidades, e neurônios conectados por uma rede de ligações sinápticas, com as respectivas intensidades (os pesos) associadas a essas sinapses.

A rede neural artificial, que corresponde à malha viária a ser percorrida pelo caixeiro, é modelada a partir de um reticulado quadrado de dimensão $L \times L$, ou um reticulado retangular de dimensão $L \times H$, conforme a instância do problema. Os N neurônios (as N cidades) que formam a rede são distribuídos no reticulado, de modo que cada célula dele pode ser ocupada por apenas um único neurônio, e inicialmente, todos os neurônios são conectados entre si. Essas conexões (esses caminhos) correspondem às sinapses existentes entre neurônios biológicos.

O conjunto de todas as ligações entre os neurônios determina a topologia de acoplamento da rede neural, no instante inicial. As conexões são consideradas não-direcionadas; ou seja, a sinapse do neurônio *a* para o neurônio *b* corresponde à mesma sinapse que liga o neurônio *b* ao neurônio *a*. E não existem sinapses que ligam um neurônio a ele próprio; isto é, a rede neural não apresenta autoconexões.

Como ponto de partida, cria-se então um espaço delimitado pelo reticulado. Nesse espaço, N cidades são distribuídas, com cada uma ligada inicialmente a todas as demais, como ilustrado na figura 3.1 para um caso de reticulado quadrado em que $L=10$ e $N=5$. Ou seja, na matriz 10×10 distribuem-se cinco cidades, existindo inicialmente caminhos conectando todos pares de cidades. Alguns casos, em que o reticulado é quadrado e a distribuição das cidades é feita de modo aleatório (como na figura 3.1), são denominados *instâncias aleatórias*. Outros casos, em que o reticulado é retangular e a distribuição das cidades é feita baseada em casos particulares publicados na biblioteca on-line conhecida como TSPLIB, são denominados *instâncias da TSPLIB* (APPLEGATE et al., 2003; REINELT, 2010). Mais a frente, o capítulo 4 traz mais detalhes.

A intensidade de cada sinapse é igual ao inverso do valor da distância euclidiana entre o par de neurônios que ela conecta. Ou seja, quanto mais próximo um neurônio está de outro ao qual está conectado, mais forte é a sinapse (maior é o peso sináptico) que os liga.

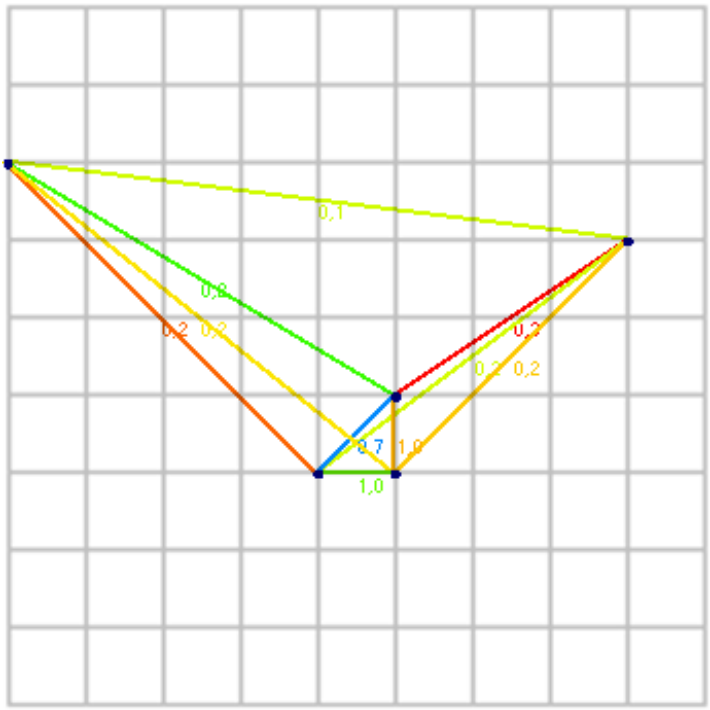

Figura 3.1. Exemplo de rede neural criada em uma matriz *10×10* na qual as posições de cinco neurônios foram sorteadas aleatoriamente. Os círculos escuros correspondem a esses neurônios e os números ao lado das conexões indicam os valores dos pesos sinápticos (esses números foram representados com apenas um algarismo significativo para facilitar a visualização). Note que os caminhos mais compridos correspondem aos pesos menores.

3.2 AS REGRAS DE MODIFICAÇÃO SINÁPTICA

O peso da sinapse entre dois neurônios sofre alteração em função das atividades desses neurônios. Esse processo de modificação sináptica obedece à regra de Hebb. Assim, há um fortalecimento da sinapse correspondente a um par de neurônios quando eles disparam simultaneamente. Em contrapartida, quando ocorre o disparo não-simultâneo ou não ocorre disparo, há então um enfraquecimento dessa sinapse. A saída y_k do neurônio k é dada pela função de ativação φ aplicada ao estímulo b_k de origem externa mais a entrada v_k decorrente da interação entre esse neurônio k e seus vizinhos. Desse modo, y_k é dado por:

$$y_k = \varphi(v_k + b_k) \qquad (3.1)$$

A entrada v_k é constituída pelo somatório de cada sinal de entrada x_k multiplicado pelo peso w_{kj} de cada sinapse, entre o neurônio k e cada um de seus $j=1,2,...,m$ vizinhos. Matematicamente, a entrada v_k é assim descrita:

$$v_k = \sum_{j=0}^{m} w_{kj}\, x_j \qquad (3.2)$$

A função de ativação φ é do tipo degrau unitário; assim, sua saída vale 0 se seu argumento $z = v_k + b_k$ é menor do que o limiar λ, e vale 1 se seu argumento é maior ou igual a λ. A equação 3.3 mostrada neste capítulo está diferente da equação 2.1, Lá, o limiar valia 0. Aqui, o algoritmo permite outros valores limiares para φ (e não apenas 0) representado então por λ. Assim:

$$\varphi(z) = \begin{cases} 1 \text{ se } z \geq \lambda \\ 0 \text{ se } z < \lambda \end{cases} \qquad (3.3)$$

O neurônio pode estar unicamente em um de três diferentes estados: repouso, ativo e refratário. O disparo do neurônio só ocorre se ele estiver no estado de repouso, com $y_k = 0$. Ao disparar, o neurônio entra no estado ativo e consequentemente sua saída se torna $y_k = 1$. Do estado ativo, o neurônio passa para o estado refratário, após um determinado número de passos de tempo, retornando então sua saída para $y_k = 0$. Apesar de apresentarem a mesma saída $y_k = 0$, os estados de repouso e refratário diferem quanto à susceptibilidade ao disparo. Enquanto o estado de repouso é susceptível ao disparo, o estado refratário não pode proporcionar disparos. A figura 3.2 mostra o diagrama de transição de estados do neurônio.

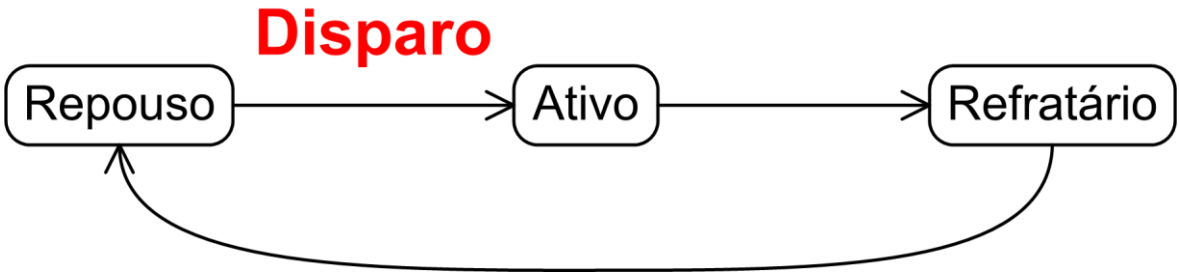

Figura 3.2: Diagrama de transição de estados do modelo de neurônio.

Para cada neurônio da rede no estado de repouso, a entrada v_k é somada ao estímulo b_k (lembrando que ambos podem ser nulos). Se o valor resultante é maior ou igual ao limiar λ, então a saída do neurônio y_k torna-se igual a *1* representando, assim, um disparo. Neurônios no estado ativo vão para o estado refratário e neurônios no estado refratário vão para o estado de repouso. Cada transição ocorre após um certo número de passos de tempo. Cada neurônio da rede é analisado aos pares, levando em conta seus respectivos vizinhos. Isto é, se ele e um vizinho disparam simultaneamente, a sinapse que liga ambos é fortalecida. Caso apenas um deles ou nenhum dispare, a sinapse sofre um enfraquecimento. A contabilização de um passo de tempo no algoritmo somente ocorre após todas as ações definidas em cada algoritmo serem executadas. Os algoritmos desenvolvidos estão descritos na seção seguinte.

3.3 OS ALGORITMOS PROPOSTOS

Como já mencionado, cada neurônio da rede neural corresponde a uma cidade a ser visitada, cada sinapse representa o caminho entre duas cidades, e o peso sináptico vale o inverso da distância entre as cidades. Dessa forma, os algoritmos foram criados para resolver o problema do caixeiro viajante simétrico. Foram elaborados ao todo 7 algoritmos, de certa forma seguindo uma linha de evolução diante de uma versão inicial, com cunho mais biológico, até uma última versão, com cunho mais computacional. O primeiro algoritmo é chamado de v0 (versão 0) e o último de v6 (versão 6). Pseudocódigos e ilustrações são mostrados no Apêndice A visando a melhor compreensão.

Os parâmetros utilizados nos algoritmos foram assim denominados: η como fator de decremento, μ como fator de incremento, λ como limiar da função de ativação, α como o número de passos de tempo do neurônio no estado ativo, β como o número de passos de tempo do neurônio no estado refratário, φ como a constante de saturação do neurônio.

No primeiro algoritmo determinista v0, a busca pela solução do problema do caixeiro viajante ocorre conforme as ações descritas abaixo, entre dois passos de tempo consecutivos:

- Todas as ligações existentes são enfraquecidas, de modo que o peso de cada uma é diminuído da constante η (pois, como os neurônios não são estimulados, as intensidades das sinapses apenas diminuem com o passar do tempo). As ligações enfraquecem até a saturação definida por $-\theta$.

- Considere a disposição espacial das cidades no plano cartesiano com *x=0* e *y=0* representando o canto esquerdo-superior, e com *x=L-1* e *y=L-1* representando o canto direito-inferior. A cidade mais próxima da origem (em *x=0* e *y=0*) é escolhida como ponto de partida. O caminho percorrido até outra cidade é aquele que passa pela ligação de maior peso entre a cidade de partida e a seguinte. Ou seja, partindo-se da cidade *c*, que possui ligações com as cidades vizinhas c_1, c_2, ..., c_n, a cidade c_j *(j=1,2,...n)* a ser visitada é aquela conectada com *c* pela ligação de maior peso sináptico. Seguindo essa abordagem "gulosa", é montado um possível caminho-solução passando por todas as cidades.
- Cria-se então uma lista em que as cidades são classificadas em ordem decrescente pela quantidade de ligações que cada uma delas possui no momento. Dessa maneira, a cidade com o maior número de ligações figura na primeira posição e a cidade com o menor número de ligações ocupa a última posição.
- Essa lista ordenada é então varrida, do topo para a base, tomando as cidades aos pares, fixando uma cidade e com ela formando um par, percorrendo as demais cidades da lista. A figura 3.3 mostra como são realizadas as interações. As ligações existentes entre cada par de cidades com valor menor ou igual a zero são eliminadas, desde que tais ligações não sejam utilizadas no caminho-solução montado anteriormente.

Essas ações são executadas repetidas vezes por determinado número de passos de tempo. Conforme o tempo passa, ligações vão sendo eliminadas, restando apenas um único caminho que percorre todas as cidades; conforme definição do problema do caixeiro viajante, ele passa apenas uma vez por cada uma delas. Tal caminho encontrado recebe o nome de caminho-solução, pois representa a solução encontrada pelo algoritmo para o problema do caixeiro viajante.

Analisando os resultados obtidos durante as simulações com o algoritmo v0, observamos que o caminho-solução encontrado em cada passo de tempo permanecia sempre o mesmo, ao longo de todos os passos daquela simulação. Esse fato pode ser explicado pelo uso de um mesmo ponto de partida, o que impede o encontro de caminhos alternativos em relação ao inicialmente estabelecido.

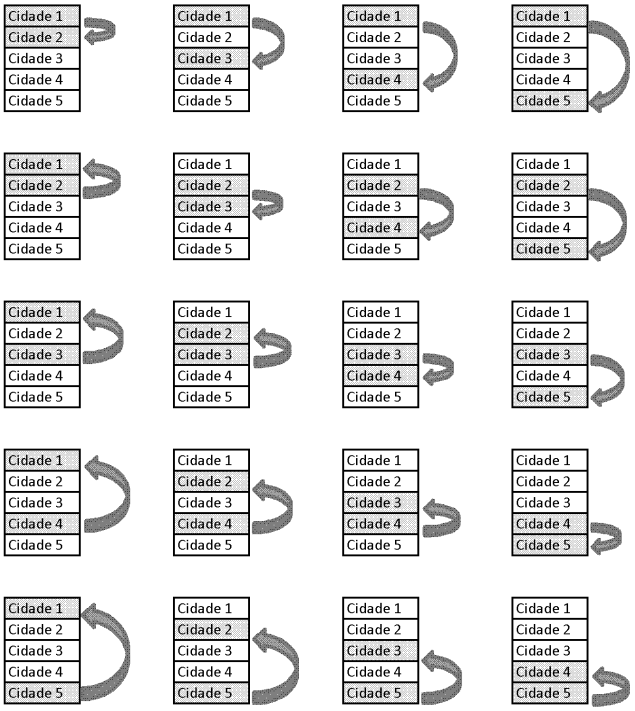

Figura 3.3. A varredura da lista ordenada é feita de cima para baixo e da esquerda para a direita, considerando pares de cidades (neurônios). Suponha que, nessa lista, a cidade com mais ligações é a 1, seguida pela 2, pela 3, pela 4 e, finalmente, pela cidade 5. Varre-se essa lista assim: fixa-se a cidade 1 e analisam-se as sinapses que ela faz com as demais cidades; depois fixa-se a cidade 2 e repete-se o procedimento. Finaliza-se a varredura analisando as sinapses da cidade 5 com as outras quatro cidades. Só são eliminadas as ligações (sinapses) com peso menor ou igual a zero e que não fazem parte do caminho-solução.

Na tentativa de buscar uma forma de explorar diferentes caminhos-solução ao longo da simulação, modificamos o algoritmo v0 para que ele fosse transformado em um algoritmo não-determinista. O resultado dessa modificação foi denominado como algoritmo v1 e pode ser descrito pelas ações definidas a seguir, também executadas entre dois passos de tempo consecutivos:

- Uma cidade é selecionada aleatoriamente para servir de ponto de partida para a criação de um possível caminho-solução.
- Partindo-se da cidade selecionada, o caminho percorrido até outra cidade é aquele que passa pela ligação de maior peso entre a cidade de partida e a seguinte, assim como ocorre no algoritmo v0 (abordagem gulosa). Ou seja, partindo-se da cidade *c*, que possui ligações com as cidades c_1, c_2, ..., c_n, a cidade c_j *(j=1,2,...n)* a ser visitada é aquela conectada com *c* pela ligação de maior peso sináptico. Seguindo essa regra, é montado um possível caminho-solução passando por todas as cidades.

Esse caminho-solução é comparado com o caminho-solução do passo de tempo anterior (exceto na primeira iteração), só sendo então adotado caso seja "melhor" (mais curto) que o anterior. Caso o caminho-solução do passo anterior seja "melhor", ele permanece como caminho-solução do passo atual.

- Como neurônios, as cidades processam suas entradas que podem receber ou não estímulos externos. Seguindo então os conceitos de plasticidade sináptica, todas as ligações existentes têm seus valores atualizados, fortalecendo ou enfraquecendo essas ligações, de modo que o peso de cada uma ou é aumentado da constante μ ou diminuído da constante η. Conforme a regra de Hebb, se dois neurônios estimulados disparam simultaneamente a ligação entre eles se fortalece, caso contrário se enfraquece. As ligações se fortalecem até a saturação definida por θ ou enfraquecem até $-\theta$. Quando estimulado, um neurônio permanece no estado ativo por α passos de tempo, e em seguida passa ao estado refratário permanecendo por β passos de tempo, antes de voltar ao estado de repouso.
- Cria-se então uma lista em que as cidades são classificadas em ordem decrescente pela quantidade de ligações que cada uma delas possui no momento. Dessa maneira, a cidade com o maior número de ligações figura na primeira posição e a cidade com o menor número de ligações ocupa a última posição.
- Essa lista ordenada é então varrida, do topo para a base, tomando as cidades aos pares, fixando uma cidade e com ela formando um par, percorrendo as demais cidades da lista. A figura 3.3 mostra como são realizadas as interações. As ligações existentes entre cada par de cidades com valor menor ou igual a zero são eliminadas, desde que tais ligações não sejam utilizadas no caminho-solução montado anteriormente. Impõe-se que as cidades unidas pela ligação a ser eliminada recebem um estímulo. Tal estímulo provoca o disparo dos neurônios correspondentes.

Assim como no algoritmo v0, no algoritmo v1 essas ações são executadas repetidas vezes por um determinado número de passos de tempo. Com o passar do tempo, as ligações vão sendo eliminadas, restando apenas o caminho-solução.

Como forma de melhorar os resultados, se aproximando mais vezes da solução ótima, o algoritmo sofreu modificações que deram origem ao algoritmo v2. Nessa versão, introduzem-se sinapses inibitórias e todos os neurônios são estimulados a cada passo de tempo. As ações realizadas entre dois passos de tempo consecutivos são explicadas a seguir:

- Uma cidade é selecionada aleatoriamente para servir de ponto de partida para a criação de um possível caminho-solução.
- Assim como ocorre nos algoritmos anteriores, partindo-se dessa cidade de forma gulosa, o caminho percorrido até outra cidade é aquele que passa pela ligação de maior peso entre a cidade de partida e a seguinte. Seguindo essa regra, é montado um possível caminho-solução passando por todas as cidades.
- As ligações que compõem o caminho-solução são eleitas como sinapses excitatórias. Todas as demais ligações existentes são então eleitas como sinapses inibitórias, independentemente do valor do peso ser positivo ou negativo.
- Todas as cidades, tratadas como neurônios, são estimuladas. Todas as ligações existentes têm seus valores atualizados, fortalecendo ou enfraquecendo essas ligações conforme o tipo de sinapse (excitatória e inibitória), de modo que o peso de cada uma ou é aumentado da constante μ ou diminuído da constante η. As ligações se fortalecem até a saturação definida por θ ou enfraquecem até $-\theta$. Ao disparar, um neurônio permanece no estado ativo por α passos de tempo, e em seguida passa ao estado refratário permanecendo por β passos de tempo, antes de voltar ao estado de repouso. As ligações existentes com valor menor ou igual a zero são eliminadas, desde que tais ligações não sejam utilizadas no caminho-solução montado anteriormente.

Na tentativa de evitar o sorteio de um mesmo neurônio como ponto de partida para a montagem do caminho-solução, o algoritmo v2 sofreu uma modificação no processo de escolha do ponto de partida. A nova versão foi denominada de algoritmo v3. A única ação entre dois passos de tempo consecutivos, que difere do algoritmo anterior, pode ser descrita como:

- Uma cidade é selecionada aleatoriamente para servir de ponto de partida para a criação de um possível caminho-solução. A cidade selecionada passa a pertencer a um conjunto de cidades sorteadas. Assim, apenas as cidades ainda não sorteadas permanecem passíveis de seleção no passo seguinte. Após todas as cidades serem sorteadas, o conjunto é esvaziado, permitindo, assim, que qualquer cidade possa ser sorteada novamente.

Ao analisar as simulações do algoritmo v3, foi possível notar que, durante sua execução, os pesos dos caminhos encontrados oscilavam. Por vezes um caminho apresentava peso maior que o caminho anterior; em outras, ele era menor. A fim de evitar que caminhos com pesos maiores (soluções melhores) fossem trocados por caminhos com pesos menores (soluções piores) ao longo da execução, foi elaborado o algoritmo v4 que conta com uma modificação assim descrita:

- Assim como ocorre nos algoritmos anteriores, partindo-se da cidade selecionada, o caminho percorrido até outra cidade é aquele que passa pela ligação de maior peso entre a cidade de partida e a seguinte. Seguindo essa regra, é montado um caminho passando por todas as cidades. Após o primeiro passo de tempo, o peso total do caminho atual (soma dos pesos de cada uma das ligações que compõem o caminho atual) é comparado ao peso total do melhor caminho (maior peso total) encontrado nos passos anteriores. Se o peso total atual for maior que o peso total do melhor caminho encontrado anteriormente, o caminho atual é considerado e armazenado como o melhor. Se o peso total atual for menor que o peso total do melhor caminho encontrado anteriormente, o caminho atual é então descartado sendo considerado como atual o melhor já existente.

Esse algoritmo v4 normalmente encontrava a solução ótima para 10 cidades num tempo razoável; mas, para um número da ordem de 20 cidades, seu tempo de processamento se tornou muito grande (da ordem de dias/semanas).

Na tentativa de criar um algoritmo capaz de resolver, num tempo de processamento "razoável", instâncias publicadas na TSPLIB, em que o número de cidades é da ordem de 100, modificou-se o algoritmo v4, o que deu origem ao algoritmo v5. Essas alterações consistem de:

- Na primeira iteração, uma cidade é selecionada aleatoriamente para servir de ponto de partida. A cidade selecionada passa a pertencer a um conjunto de cidades selecionadas como iniciais. Nas iterações seguintes, a cidade escolhida é a mais próxima da cidade selecionada no passo de tempo anterior que ainda não se encontra no conjunto de cidades selecionadas como iniciais. Após todas as cidades terem sido selecionadas, o conjunto é esvaziado, permitindo assim que todo o ciclo se repita, do sorteio da primeira cidade à escolha das seguintes.

- A partir da cidade inicialmente selecionada, as duas cidades mais próximas ainda não visitadas são escolhidas como cidades que comporão o caminho. A ligação de maior peso (que se liga à primeira cidade) é denominada de “lado esquerdo” do caminho, e a com menor peso (que liga a inicial à segunda cidade) é denominada de “lado direito” do caminho. Tal configuração pode ser imaginada como um “V”, sendo o vértice a cidade inicial, a extremidade da esquerda a cidade “lado esquerdo” e a extremidade da direita a cidade “lado direito”. Busca-se então a quarta cidade, partindo daquela chamada de “lado esquerdo”. Essa quarta cidade é encontrada via busca gulosa; ou seja, ela é a que possui a ligação com maior peso que a conecta à cidade “lado esquerdo”. Esse processo se repete até fechar o caminho. Na próxima iteração, a cidade “lado esquerdo” da iteração anterior é a inicial. A cidade “lado esquerdo” desta iteração deve ser a mais próxima que não tenha sido selecionada como inicial nas iterações anteriores. A cidade “lado direito” desta iteração é a mais próxima, sem a restrição de não ter sido selecionada como cidade inicial. A partir do “lado” que tiver o maior peso, faz-se uma busca gulosa, até fechar o caminho. Esse processo se repete até o conjunto de cidades selecionadas como iniciais ser totalmente preenchido, quando então todo o processo se repete conforme a primeira iteração.

Notando que, no caminho encontrado em cada iteração, a ligação entre a última cidade e a cidade inicial (a ligação que torna o percurso fechado, aquela que fecha o caminho) costumava ser demasiadamente fraca em relação às demais ligações (ou seja, tinha uma distância “muito grande”), o algoritmo v5 foi modificado para não selecionar, como ponto de partida, as cidades conectadas por essa ligação. A idéia era impedir que a construção do caminho se iniciasse por essas cidades nas iterações seguintes, a fim de que o novo caminho encontrado não contivesse novamente a mesma ligação fraca. O algoritmo v6, sendo o último apresentado neste trabalho, pode ser descrito então pelas seguintes ações realizadas entre dois passos de tempo consecutivos:

- Na primeira iteração, uma cidade é selecionada aleatoriamente para servir de ponto de partida. A cidade selecionada passa a pertencer a um conjunto de cidades selecionadas como iniciais. Nas iterações seguintes, a cidade escolhida é a mais próxima da cidade selecionada no passo de tempo anterior que ainda não se encontra no conjunto de cidades selecionadas como iniciais. Após todas as cidades

terem sido selecionadas, o conjunto é esvaziado, permitindo assim que o ciclo se repita.

- A partir da cidade inicialmente selecionada, as duas cidades mais próximas ainda não visitadas são escolhidas como cidades que comporão o caminho. A ligação de maior peso (que se liga à primeira cidade) é denominada de "lado esquerdo" do caminho, e a com menor peso (que liga a inicial à segunda cidade) é denominada de "lado direito" do caminho. Tal configuração pode ser imaginada como um "V", sendo o vértice a cidade inicial, a extremidade da esquerda a cidade "lado esquerdo" e a extremidade da direita a cidade "lado direito". Busca-se então a quarta cidade, partindo daquela chamada de "lado esquerdo". Essa quarta cidade é encontrada via busca gulosa; ou seja, ela é a que possui a ligação com maior peso que a conecta à cidade "lado esquerdo". Esse processo se repete até fechar o caminho. Na próxima iteração, a cidade "lado esquerdo" da iteração anterior é a inicial. A cidade "lado esquerdo" desta iteração deve ser a mais próxima que não tenha sido selecionada como inicial nas iterações anteriores. A cidade "lado direito" desta iteração é a mais próxima, sem a restrição de não ter sido selecionada como cidade inicial. A partir do "lado" que tiver o maior peso, faz-se uma busca gulosa, até fechar o caminho. Esse processo se repete até o conjunto de cidades selecionadas como iniciais ser totalmente preenchido, quando então todo o processo se repete conforme a primeira iteração.
- A cada iteração, marca-se a ligação mais fraca. As duas cidades conectadas por essa ligação não são escolhidas como cidade inicial nas iterações seguintes, até o conjunto de cidades selecionadas como iniciais ser esvaziado.
- As ligações que compõem o caminho-solução são eleitas como sinapses excitatórias. Todas as demais ligações existentes são então eleitas como sinapses inibitórias, independentemente do valor do peso ser positivo ou negativo.
- Todas as cidades, tratadas como neurônios, são estimuladas. Todas as ligações existentes têm seus valores atualizados, fortalecendo ou enfraquecendo essas ligações conforme o tipo de sinapse (excitatória e inibitória), de modo que o peso de cada uma ou é aumentado da constante μ ou diminuído da constante η. As ligações se fortalecem até a saturação definida por θ ou enfraquecem até $-\theta$. Ao disparar, um neurônio permanece no estado ativo por α passos, e em seguida passa ao estado refratário permanecendo por β passos de tempo, antes de voltar ao estado

de repouso. As ligações existentes com valor menor ou igual a zero são eliminadas, desde que tais ligações não sejam utilizadas no caminho-solução montado anteriormente.

No capítulo seguinte, mostram-se os resultados dos experimentos numéricos realizados com esses algoritmos.

4 EXPERIMENTOS NUMÉRICOS

4.1 INTRODUÇÃO

As simulações foram realizadas em um computador pessoal equipado com processador *Intel Core i5* de *2.66 GHz, 4.0 Gb* de memória *RAM*, sistema operacional *Microsoft Windows 7 Ultimate*. Os algoritmos foram desenvolvidos utilizando a linguagem de programação *C#*, através da ferramenta *Microsoft Visual Studio 2008*, com a plataforma de desenvolvimento *.NET Framework v2.0*.

Vale ressaltar que as figuras ilustrando a rede nesse capítulo possuem intersecções de ligações apenas visuais, ou seja, as linhas foram traçadas ligando uma cidade à outra sem um tratamento visual específico para sobreposição dos traços, e não representam soluções inválidas para o problema.

4.2 INSTÂNCIAS GERADAS ALEATORIAMENTE

Cada simulação realizada corresponde a uma nova disposição espacial das cidades a serem visitadas. No caso em que a disposição foi gerada aleatoriamente, foram determinados todos os caminhos possíveis, a fim de se encontrar a solução ótima e compará-la com o comprimento dos caminhos-solução encontrados pelos algoritmos. Para determinar esse comprimento, os pesos das ligações entre cada par de cidades que formam esses caminhos foram somados, a fim de se calcular o peso total *W*. Vale novamente ressaltar que o número de caminhos distintos possíveis vale *S=(N-1)!* e que quanto maior *W* mais curto é o caminho.

Os caminhos possíveis foram ordenados de forma decrescente pelo valor de *W*, de tal forma que o melhor (mais curto) caminho aparece na posição 1 e o pior (mais comprido) caminho ocupa a última posição. O caminho-solução gerado pelo algoritmo foi então localizado nessa lista. O desempenho do algoritmo foi avaliado quantitativamente através da posição ocupada pelo caminho-solução na lista ordenada dos caminhos possíveis.

Para avaliar o desempenho dos algoritmos ao longo das suas modificações bem como o efeito das variações nos valores dos parâmetros, foram realizados, ao todo, *18* experimentos. Cada experimento foi composto por *200* simulações com *N=10*. No caso, existem *S=362880* caminhos fechados possíveis para cada simulação em particular (cada disposição espacial de *10* cidades). Para cada simulação realizada, o algoritmo foi executado por *1000* passos de

tempo. A figura 4.1 mostra a topologia inicial de um caso para *N=10* a ser resolvido (lado esquerdo), e o caminho-solução (que é o ótimo) encontrado (lado direito).

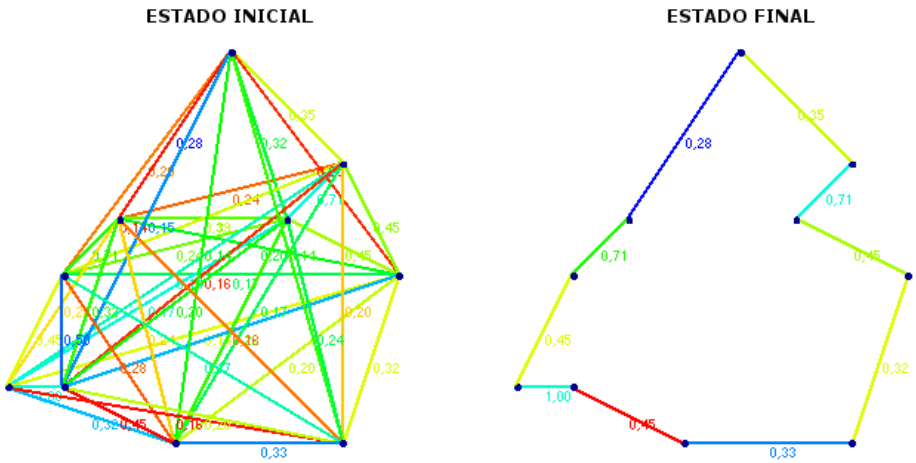

Figura 4.1. Exemplo de rede *N=10* em seu estado inicial no lado esquerdo e em um estado final no lado direito mostrando o caminho-solução encontrado pelo algoritmo v0.

Cada experimento realizado foi denominado E1, E2, E3 e assim sucessivamente, até E18. O primeiro experimento E1 foi realizado com o algoritmo v0. Os experimentos E2 a E13 foram realizados com o algoritmo v1. O experimento E14 foi realizado com o algoritmo v2; o E15, com o algoritmo v3; o E16, com o algoritmo v4; o E17, com o algoritmo v5 e, finalmente, o E18, com o algoritmo v6. Resultados preliminares com as versões v0 e v1 já foram publicados em anais de congresso nacional (veja apêndice B).

As tabelas 4.1 e 4.2 apresentam os valores dos parâmetros utilizados nos 18 experimentos juntamente com a respectiva versão do algoritmo. Note que a versão v1 foi a mais explorada.

Tabela 4.1. Valores dos parâmetros utilizados nos experimentos E1 a E9. Os valores destacados se referem a alterações em relação ao experimento anterior.

Parâmetro	**E1 v0**	**E2 v1**	**E3 v1**	**E4 v1**	**E5 v1**	**E6 v1**	**E7 v1**	**E8 v1**	**E9 v1**
Número de neurônios	*N=10*	*N=10*	*N=10*	*N=10*	*N=10*	*N=10*	*N=10*	*N=10*	*N=10*
Dimensão da matriz	*L=10*	*L=10*	*L=10*	*L=10*	*L=10*	*L=10*	*L=10*	*L=10*	*L=10*
Fator de decremento	*η=0,01*	*η=0,01*	***η=0,1***	*η=0,01*	*η=0,01*	***η=0,05***	***η=0,01***	*η=0,01*	*η=0,01*
Fator de incremento	*μ=0,2*	*μ=0,02*	*μ=0,02*	***μ=0,2***	***μ=0,01***	***μ=0,02***	***μ=0,1***	***μ=0,02***	*μ=0,02*
Limiar da função de ativação	*λ=2,0*	*λ=1,0*	*λ=1,0*	*λ=1,0*	*λ=1,0*	*λ=1,0*	*λ=1,0*	*λ=1,0*	*λ=1,0*
Passos no estado ativo	*α=1*	*α=1*	*α=1*	*α=1*	*α=1*	*α=1*	*α=1*	*α=1*	***α=2***
Passos no estado refratário	*β=2*	*β=2*	*β=2*	*β=2*	*β=2*	*β=2*	*β=2*	***β=3***	*β=3*
Constante de saturação	*φ=2,0*	*φ=1,0*	*φ=1,0*	*φ=1,0*	*φ=1,0*	*φ=1,0*	*φ=1,0*	*φ=1,0*	*φ=1,0*

Tabela 4.2. Valores dos parâmetros utilizados nos experimentos E10 a E18. Os valores destacados se referem a alterações em relação ao experimento anterior.

Parâmetro	E10 v1	E11 v1	E12 v1	E13 v1	E14 v2	E15 v3	E16 v4	E17 v5	E18 v6
Número de neurônios	$N=10$	$N=10$	$N=10$	$N=10$	$N=10$	$N=10$	$N=10$	$N=10$	$N=10$
Dimensão da matriz	$L=10$	$L=10$	$L=10$	$L=10$	$L=10$	$L=10$	$L=10$	$L=10$	$L=10$
Fator de decremento	$\eta=0,01$	$\eta=0,01$	$\eta=0,01$	$\eta=0,01$	$\eta=0,01$	$\eta=0,01$	$\eta=0,01$	$\eta=0,01$	$\eta=0,01$
Fator de incremento	$\mu=0,02$	$\mu=0,02$	$\mu=0,02$	$\mu=0,02$	$\mu=0,02$	$\mu=0,02$	$\mu=0,02$	$\mu=0,02$	$\mu=0,02$
Limiar da função de ativação	$\lambda=1,0$	$\lambda=1,0$	$\lambda=1,0$	$\lambda=1,0$	$\lambda=1,0$	$\lambda=1,0$	$\lambda=1,0$	$\lambda=1,0$	$\lambda=1,0$
Passos no estado ativo	**$\alpha=1$**	**$\alpha=3$**	$\alpha=3$	**$\alpha=2$**	**$\alpha=3$**	$\alpha=3$	$\alpha=3$	$\alpha=3$	$\alpha=3$
Passos no estado refratário	**$\beta=1$**	**$\beta=3$**	**$\beta=4$**	**$\beta=5$**	**$\beta=4$**	$\beta=4$	$\beta=4$	$\beta=4$	$\beta=4$
Constante de saturação	$\varphi=1,0$	$\varphi=1,0$	$\varphi=1,0$	$\varphi=1,0$	$\varphi=1,0$	$\varphi=1,0$	$\varphi=1,0$	$\varphi=1,0$	$\varphi=1,0$

Os resultados dos experimentos foram agrupados para melhor visualização mais adiante, nas tabelas 4.3 e 4.4. No experimento E1, o algoritmo v0 puramente determinista encontrou a solução ótima em *21,5%* dos casos. Nos demais *87,5%*, ele encontrou soluções localizadas dentre as *100* primeiras posições na lista de caminhos ordenados dos mais curtos aos mais longos, de um total de *362880* posições possíveis.

No experimento E2, já com a primeira versão de um algoritmo não-determinista, o algoritmo v1 encontrou a solução-ótima em *79,0%* dos casos (contra *21,5%* do algoritmo v0 no experimento E1) e em *21,0%* dos casos restantes apresentaram soluções até a 100^a posição na lista de caminhos ordenados dos mais curtos aos mais longos (contra *87,5%* do algoritmo v0 no experimento E1).

Depois de constatada a sensível melhora do desempenho, obtida através do algoritmo v1, novos experimentos foram elaborados com esse algoritmo, na tentativa de encontrar os valores de parâmetros mais adequados à resolução desse tipo de problema, com $L=10$ e $N=10$. Assim, no experimento E3, o fator de decremento foi aumentado para $\eta=0,1$ com os demais parâmetros mantidos com os valores do experimento anterior (alteração em destaque na tabela 4.1). Os resultados mostraram uma degradação em relação aos resultados apresentados pelo experimento E2, com menos ocorrências de soluções ótimas *36,5%* (contra *79,0%* no experimento E2).

Em seguida, no experimento E4, o fator de decremento foi restaurado para o valor do experimento E2 com $\eta=0,01$. Já o fator de incremento foi aumentado para $\eta=0,2$ sendo que os demais parâmetros foram mantidos com os mesmos valores que no experimento E2. Os resultados do experimento E4 mostraram números semelhantes aos números do experimento E2, indicando que a elevação do fator de incremento parece pouco influenciar os resultados.

No experimento E5, o fator de incremento tem o mesmo valor do fator de decremento, ou seja, com $\eta=\mu=0,01$. Os demais parâmetros foram mantidos com os mesmos valores do experimento E2. Os resultados do experimento E5 mostraram números bem próximos aos números do experimento E2, indicando que a redução do valor do fator de incremento e igualdade com o valor do fator de decremento também parece pouco influenciar os resultados.

Um novo experimento E6 foi elaborado com o fator de decremento aumentado para $\eta=0,05$ e o fator de incremento restaurado para o mesmo valor do experimento E2 com $\mu=0,02$. Os demais parâmetros foram mantidos com os mesmos valores do experimento E2. Os resultados do experimento E6 mostraram também números bem próximos aos números do experimento E2, indicando que o aumento do valor do fator de decremento parece provocar pouca alteração nos resultados.

Outro experimento E7 foi elaborado, agora com o fator de decremento restaurado para o mesmo valor usado no experimento E2 ($\eta=0,01$) e o fator de incremento aumentado para $\mu=0,1$. Os resultados desse experimento mostraram uma leve degradação em relação aos números do experimento E2, com uma redução de soluções ótimas encontradas (*72,5%* contra *79,0%* no experimento E2).

Os experimentos E2 ao E7 basearem-se na alteração dos fatores de incremento (μ) e decremento (η). Contudo, no algoritmo v1, o fator de decremento sempre é aplicado quando não há disparo, partindo da ideia de eliminar as sinapses com o passar do tempo e manter apenas um único caminho no final. Assim, aparentemente a alteração desse fator afeta a velocidade com que o algoritmo elimina as sinapses, já que ele influencia na "qualidade" do caminho encontrado: ao que tudo indica, quanto mais rápido as sinapses são eliminadas, menor a chance de outro caminho "melhor" ser encontrado.

O fator de incremento (μ) parece estimular a escolha de novos trechos de caminho, já que conforme implementado no algoritmo v1, quando uma sinapse é eliminada, os neurônios que ela ligava são estimulados para dispararem, permitindo o incremento dos pesos de outras sinapses ligadas a esses neurônios (supostamente formando trechos de caminho alternativos em relação a um trecho que passava pela sinapse sendo eliminada). O aumento do número de passos de tempo do neurônio no estado ativo tende a favorecer o fortalecimento de sinapses, pois ao permanecer mais tempo no estado ativo, um par de neurônios tende a disparar em conjunto com mais pares de neurônios em determinados momentos. Lembre que um par é estimulado a disparar somente quando a sinapse que os liga é eliminada. E se apenas um par dispara, não há consequencias, pois a sinapse que ligava esses neurônios não existe mais. O cuidado na escolha de um "bom" valor de número de passos de tempo no estado ativo (e em

contrapartida do estado refratário) passa pela ideia de não provocar uma reação de disparos em cadeia, fazendo com que toda a rede torne-se ativa, não contribuindo assim em coisa alguma para a seleção de novos trechos de caminho. Baseado nisso, os experimentos E8 a E13 foram então elaborados, com o objetivo de analisar os efeitos da alteração dos números de passos de tempo no estado ativo (α) e refratário (β).

Com base nos valores do experimento E2, é possível observar uma razão de 1:2 de número de passos de tempo no estado ativo/refratário, ou seja, $\alpha/\beta=1/2=0,5$. A idéia no experimento E8 foi tomar $\beta=3$, alterando então essa razão α/β para $1/3=0,33$ (alteração em destaque na tabela 4.1). Os demais parâmetros foram mantidos com os mesmos valores do experimento E2. Os resultados do experimento E8 aparentemente mostram uma leve degradação em relação aos números do experimento E2, com uma redução de soluções ótimas encontradas (*71,5%* contra *79,0%* no experimento E2).

No experimento E9 foi definido $\alpha=2$ e $\beta=3$, ou seja, aumentou-se a razão entre o número de passos de tempo no estado ativo e o número de passos no estado refratário para 2:3 ou $\alpha/\beta=2/3=0,6$. Os resultados do experimento E9 indicam uma aparente melhora em relação aos números do experimento E2, apresentando *81,0%* de soluções ótimas encontradas (contra *79,0%* no experimento E2).

Para o experimento E10 foram definidos valores iguais com $\alpha=\beta=1$; ou seja, com a razão de 1:1 ou $\alpha/\beta=1/1=1,0$ (alteração em destaque na tabela 4.2). Os demais parâmetros têm os valores do experimento E2. Os resultados do experimento E10 atingiram valores muito próximos aos números do experimento E2, apresentando igualmente *79,0%* de soluções ótimas encontradas e diferindo apenas nas soluções encontradas em posições inferiores na lista de caminhos ordenados dos mais curtos aos mais longos (até a 15^a posição).

Já o experimento E11 também foi realizado com a razão do número de passos de tempo no estado ativo pelo número de passos de tempo no estado refratário de 1, só que com valores $\alpha=\beta=3$; ou seja, com uma razão de 3:3, ou $\alpha/\beta=3/3=1,0$ (alteração em destaque na tabela 4.2). Os demais parâmetros têm os mesmos valores do experimento E2. Os resultados do experimento E11 atingiram valores muito próximos aos números do experimento E10 e do experimento E2, aparentemente sugerindo uma correlação dos resultados com a razão entre α e β ao apresentar *78,0%* de soluções ótimas encontradas (contra *79,0%* nos experimentos E2 e E10).

O experimento E12 foi feito com $\alpha=3$ e $\beta=4$, ou seja, com uma razão de 3:4 ou $\alpha/\beta=3/4=0,75$. Os resultados do experimento E12 mostraram valores ainda melhores que os

números do experimento E9 (até então o melhor resultado), apresentando *82,5%* de soluções ótimas encontradas (contra *81,0%* no experimento E9).

O experimento E13 foi realizado com $\alpha=2$ e $\beta=5$. Os resultados do experimento E13 mostraram uma degradação em relação aos números do experimento E2, apresentando *73,5%* de soluções ótimas encontradas.

Infelizmente, a versão v1 revelou-se incapaz de apresentar uma solução em tempo "razoável" (para nós, da ordem de horas em computador pessoal) para instâncias mais complexas ($N>20$) do problema do caixeiro viajante. Por isso, novas versões foram desenvolvidas. Os experimentos E14 a E18 foram realizados com as versões v2, v3, v4, v5 e v6.

Nos experimentos E14 a E18, utilizaram-se os mesmos valores de parâmetros do experimento E12, valores esses que haviam levado ao melhor resultado até então. Nos experimentos E14, E15 e E18 é possível notar uma degradação em relação aos últimos resultados obtidos com a versão v1. Já o experimento E16 apresentou o melhor resultado dentre os demais, atingindo *85,5%* de soluções ótimas (contra *82,5%* no experimento E12). E, finalmente, o experimento E17 mostrou resultados próximos dos últimos resultados obtidos com a versão v1.

Os resultados dos experimentos E1 a E18 foram divididos em duas tabelas para melhor apresentação. Os resultados dos experimentos E1 a E9 estão na tabela 4.3 e os dos experimentos E10 a E18, na tabela 4.4. Cada célula indica o número de vezes (em porcentagem) com que o algoritmo encontrou uma solução cuja posição é indicada na primeira coluna da tabela. Por exemplo, o algoritmo v0, no experimento E1, encontrou a solução-ótima (a solução na posição 1 na lista dos caminhos mais curtos) em *21,5%* das vezes; uma solução entre as posições 2 e 5, em *17%* das vezes, e assim por diante. Cada coluna soma, obviamente, *100%*.

Após a versão v1, a evolução do algoritmo foi guiada pela necessidade de se resolver casos com $N>20$ num tempo de processamento "razoável". Na próxima seção, apresentam-se os resultados da versão v6 aplicada a instâncias com N da ordem de *100*. Observe que, apesar de seu desempenho para instâncias aleatórias com $N=10$ ser ruim (conforme mostrado na tabela 4.4), essa versão foi a que permitiu menores tempos de execução para instâncias da TSPLIB, ao convergir mais rapidamente para um caminho-solução.

Tabela 4.3. Resultados dos experimentos E1 a E9.

Classificação	E1 v0	E2 v1	E3 v1	E4 v1	E5 v1	E6 v1	E7 v1	E8 v1	E9 v1
1 (ótimo)	21,5%	79,0%	36,5%	79,0%	78,5%	78,5%	72,5%	71,5%	81,0%
2 a 5	17,0%	15,5%	27,0%	13,0%	16,0%	14,5%	22,0%	19,5%	15,5%
6 a 15	21,0%	4,0%	17,0%	6,5%	3,5%	5,5%	5,0%	8,0%	3,0%
16 a 100	28,0%	1,5%	17,5%	1,5%	2,0%	1,5%	0,5%	1,0%	0,5%
101 a 1000	11,5%	0,0%	2,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
1001 a 10000	1,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
10001 e abaixo	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%

Tabela 4.4. Resultados dos experimentos E10 a E18.

Parâmetro	E10 v1	E11 v1	E12 v1	E13 v1	E14 v2	E15 v3	E16 v4	E17 v5	E18 v6
1 (ótimo)	79,0%	78,0%	82,5%	73,5%	49,5%	44,5%	85,5%	79,5%	39,5%
2 a 5	16,0%	17,5%	11,5%	13,0%	15,5%	22,5%	12,0%	16,5%	17,5%
6 a 15	5,0%	3,0%	4,5%	10,0%	21,5%	13,5%	2,0%	3,0%	14,5%
16 a 100	0,0%	1,5%	1,5%	3,5%	10,5%	14,5%	0,5%	1,0%	15,5%
101 a 1000	0,0%	0,0%	0,0%	0,0%	3,0%	5,0%	0,0%	0,0%	11,0%
1001 a 10000	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	2,0%
10001 e abaixo	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%

4.3 INSTÂNCIAS DA TSPLIB

Para avaliar a eficácia do algoritmo v6 diante de situações mais complexas do problema do caixeiro viajante, selecionamos 10 instâncias conhecidas do meio científico encontradas na TSPLIB (REINELT, 2010). Dentre os casos disponíveis do problema do caixeiro viajante simétrico, com representação das coordenadas como sendo do tipo euclidiana em duas dimensões, e que possuem soluções ótimas conhecidas, foram escolhidas as instâncias: a280, berlin52, ch150, eil101, eil51, eil76, kroC100, lin105, pcb442, e tsp225. Para cada instância, uma matriz de diferentes dimensões foi criada, de acordo com as coordenadas dos pontos das cidades no espaço. Cada uma das dimensões da matriz é definida conforme o valor máximo encontrado no conjunto de todas as coordenadas, tanto para X quanto para Y, sendo a largura L da matriz igual ao valor máximo de X (+10 para comodidade visual) e H a altura da matriz igual ao valor máximo de Y (+10 para comodidade visual). Ou seja, as dimensões da matriz são dadas por $L=X_{máximo}+10$ e $H=Y_{máximo}+10$.

Como nos experimentos com instâncias geradas aleatoriamente, o peso da ligação entre cada par de cidades foi definido como o inverso da distância euclidiana entre as duas

cidades. Assim, quanto mais próximo uma cidade está de outra, maior o peso (e mais próximo de 1). Por outro lado, quanto mais uma cidade dista de outra, menor é o peso (e mais próximo de 0). Com isso, ao se construir um caminho, os pesos das ligações que o formam são somados, dando origem ao peso total do caminho *W*. Vale ressaltar então que quanto maior *W* mais curto é o caminho.

Para avaliar o peso total do caminho-solução encontrado, foi definida a razão entre ele e o peso total do caminho da solução ótima, razão essa que em seu máximo pode chegar a *100%* do peso total do caminho da solução ótima. Ou seja, dados o peso total $W_{solução}$ de um caminho-solução encontrado e o peso total $W_{ótimo}$ do caminho ótimo, a qualidade da solução $Q_{solução}$ é dada pela razão $(W_{solução})/(W_{ótimo})$ (que expressa o percentual da solução encontrada em relação à solução ótima).

Como o algoritmo v6 é não-determinista, cada simulação corresponde à execução do algoritmo por *n* passos de tempo até que ele convirja para um caminho-solução. Consequentemente, como cada simulação pode gerar um caminho-solução em um tempo diferente, foram realizadas 100 simulações. Dentre os caminhos-solução encontrados e seus respectivos pesos totais, o caminho-solução de maior peso total (melhor solução) é destacado. O caminho-solução destacado, considerado o melhor em 100 simulações, é então comparado à solução ótima.

O algoritmo, desde suas versões iniciais, não permite autoconexão; ou seja, do ponto de vista neurológico, um neurônio não pode ter uma sinapse que ligue ele a ele mesmo. Mas a existência de duas cidades em um mesmo ponto geográfico pode ser considerada uma autoconexão. Por conta disso, a instância a280 teve de sofrer uma pequena alteração: como ela possui duas cidades distintas em um mesmo ponto geográfico (cidades 171 e 172), uma delas foi definida ligeiramente abaixo da outra. No caso, a cidade 172 passou das mesmas coordenadas da cidade 171 (80, 25) para as coordenadas (80, 26). A figura 4.2 mostra o mapa das cidades da instância a280 criado com dimensões *L=298* e *H=179*.

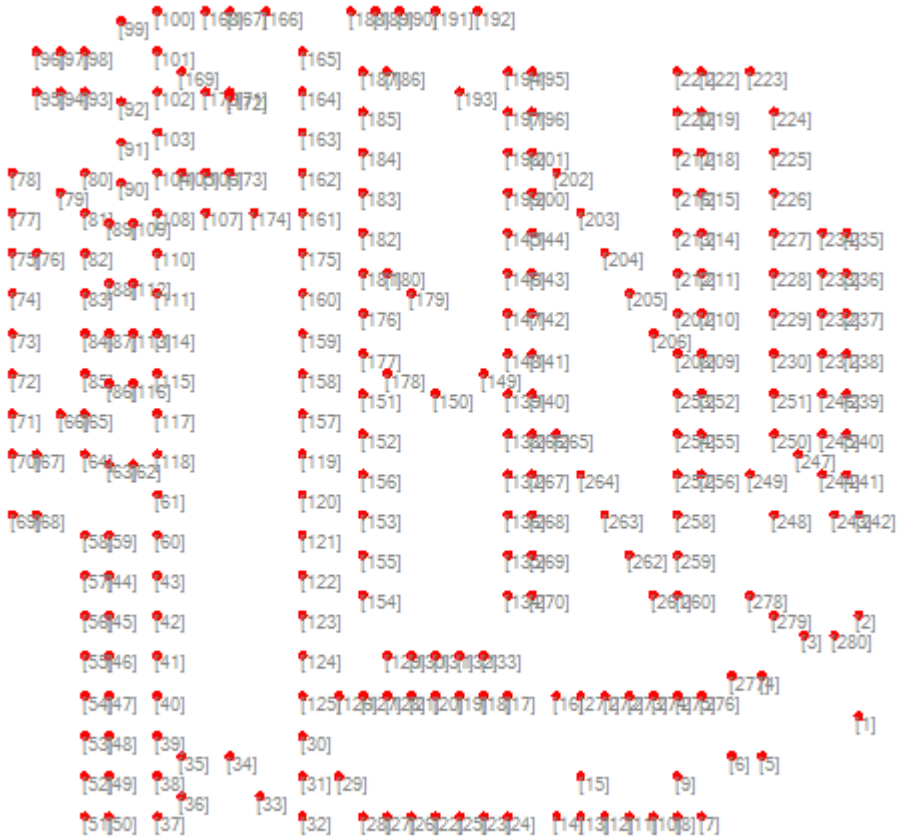

Figura 4.2. Representação das 280 cidades da instância a280 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância a280 tem peso total de aproximadamente *32,7*. O caminho da solução ótima pode ser visualizado na figura 4.3.

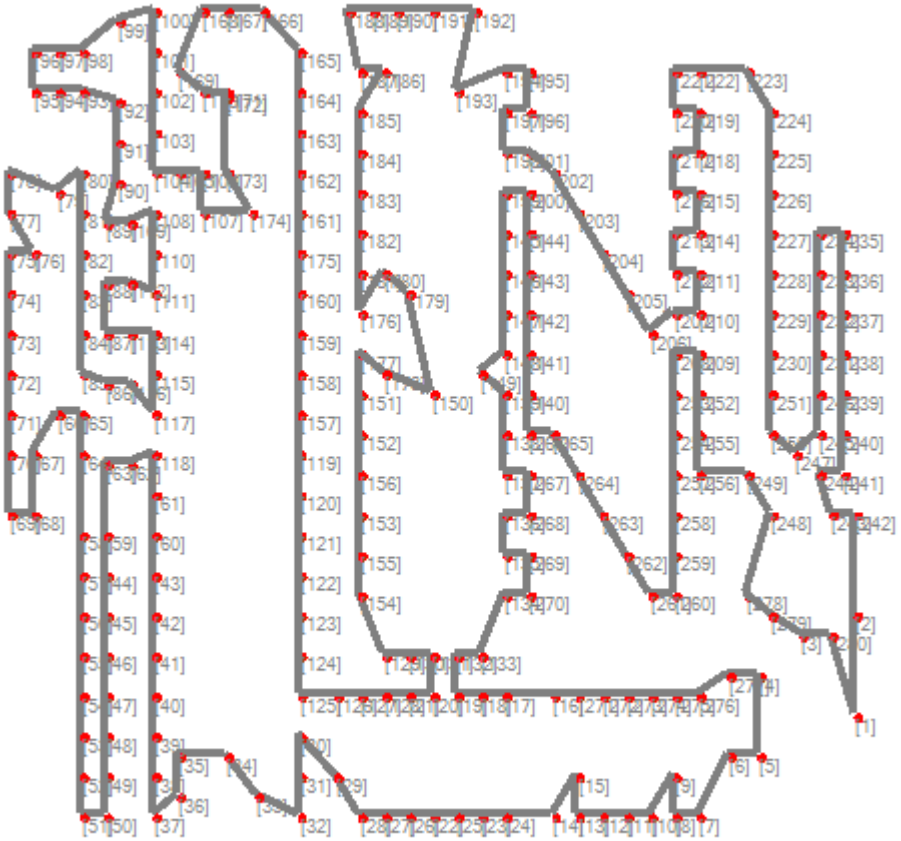

Figura 4.3. Representação do caminho da solução ótima da instância a280 com a identificação de cada uma das cidades entre colchetes.

Como a instância a280 possui *280* cidades, existem aproximadamente $5{,}99 \times 10^{562}$ caminhos possíveis. O caminho-solução apresentado pelo algoritmo para essa instância, em uma das *100* simulações, surgiu após *236393* passos de tempo em aproximadamente *1h53min43s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *97,44%* do seu peso total em relação ao peso total do caminho da solução ótima. O caminho-solução encontrado pelo algoritmo pode ser observado na figura 4.4.

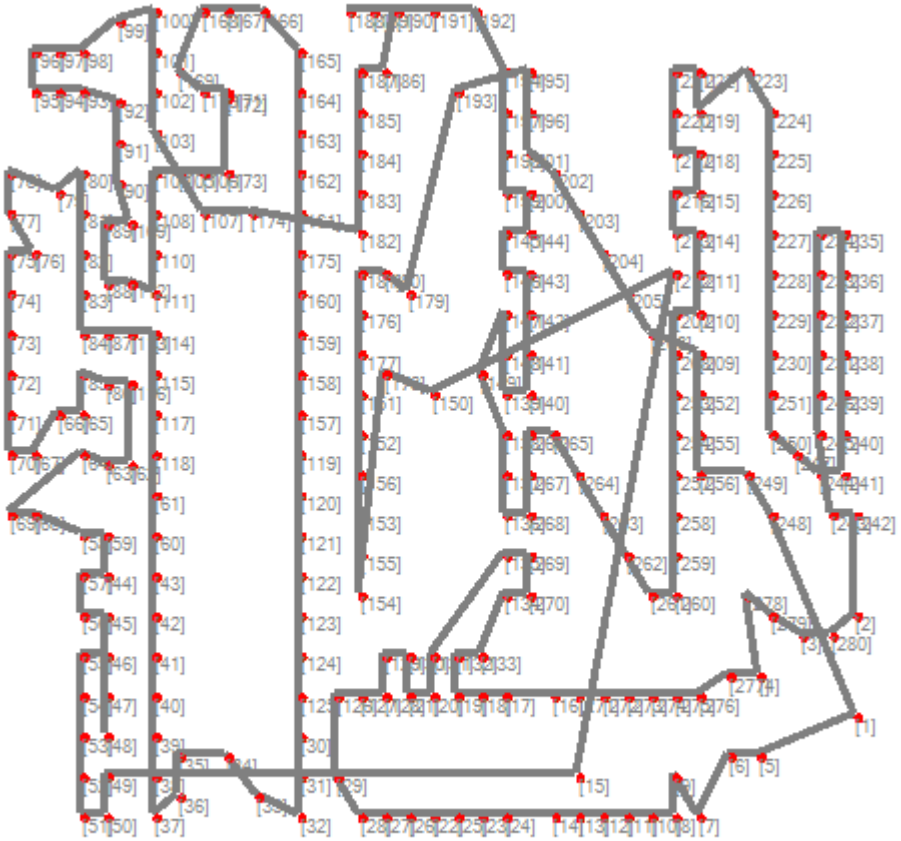

Figura 4.4. Representação do caminho-solução encontrado pelo algoritmo para a instância a280 com a identificação de cada uma das cidades entre colchetes.

O algoritmo v6 foi também testado com a instância berlin52, que pode ser visualizada na figura 4.5, com uma matriz de dimensões *L=1750* e *H=1185*.

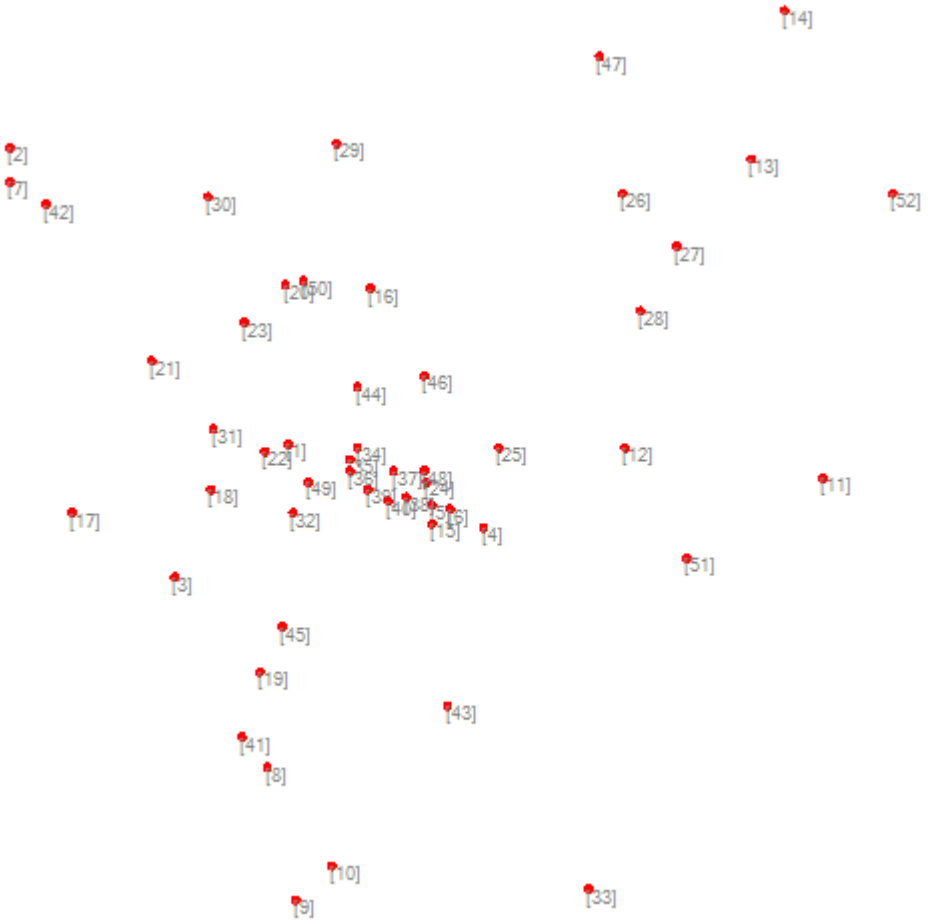

Figura 4.5. Representação das 52 cidades da instância berlin52 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância berlin52 tem peso total de aproximadamente *0,75*. O caminho da solução ótima é mostrado na figura 4.6.

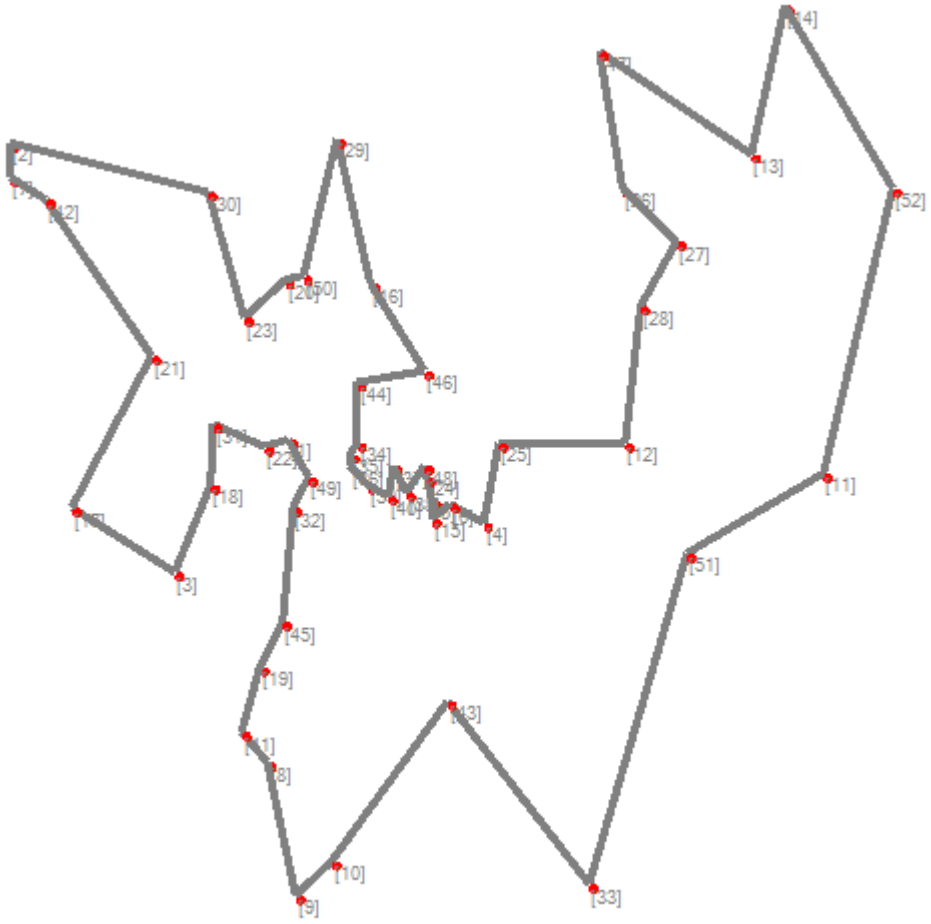

Figura 4.6. Representação do caminho da solução ótima da instância berlin52 com a identificação de cada uma das cidades entre colchetes.

No caso da instância berlin52 que possui *52* cidades, existem aproximadamente $1{,}55 \times 10^{66}$ caminhos possíveis. O caminho-solução encontrado pelo algoritmo para essa instância surgiu após *27446* passos de tempo em aproximadamente *21min15s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *99,71%* do seu peso total em relação ao peso total do caminho da solução ótima. Através da figura 4.7 é possível visualizar o caminho-solução encontrado pelo algoritmo.

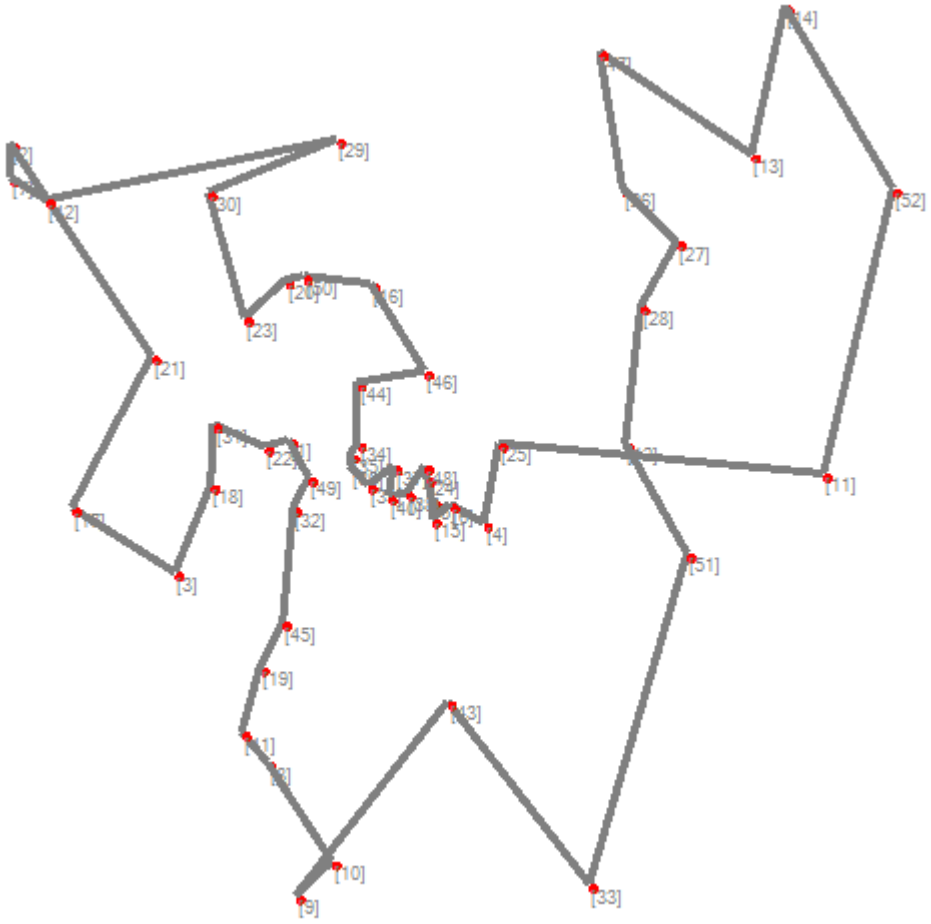

Figura 4.7. Representação do caminho-solução encontrado pelo algoritmo para a instância berlin52 com a identificação de cada uma das cidades entre colchetes.

Outra instância utilizada nos testes foi a ch150 que pode ser observada na figura 4.8. A instância foi modelada através de uma matriz de dimensões *L=707* e *H=710*.

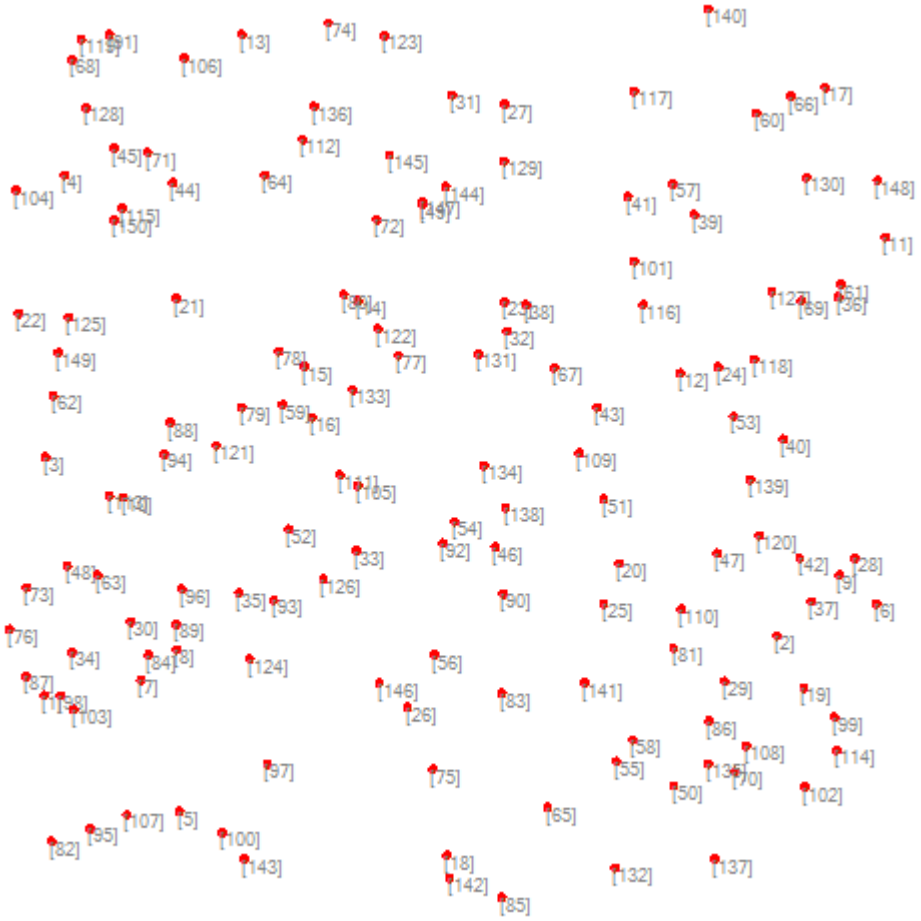

Figura 4.8. Representação das 150 cidades da instância ch150 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância ch150 possui peso total de aproximadamente *4,81*. O caminho da solução ótima está ilustrado na figura 4.9.

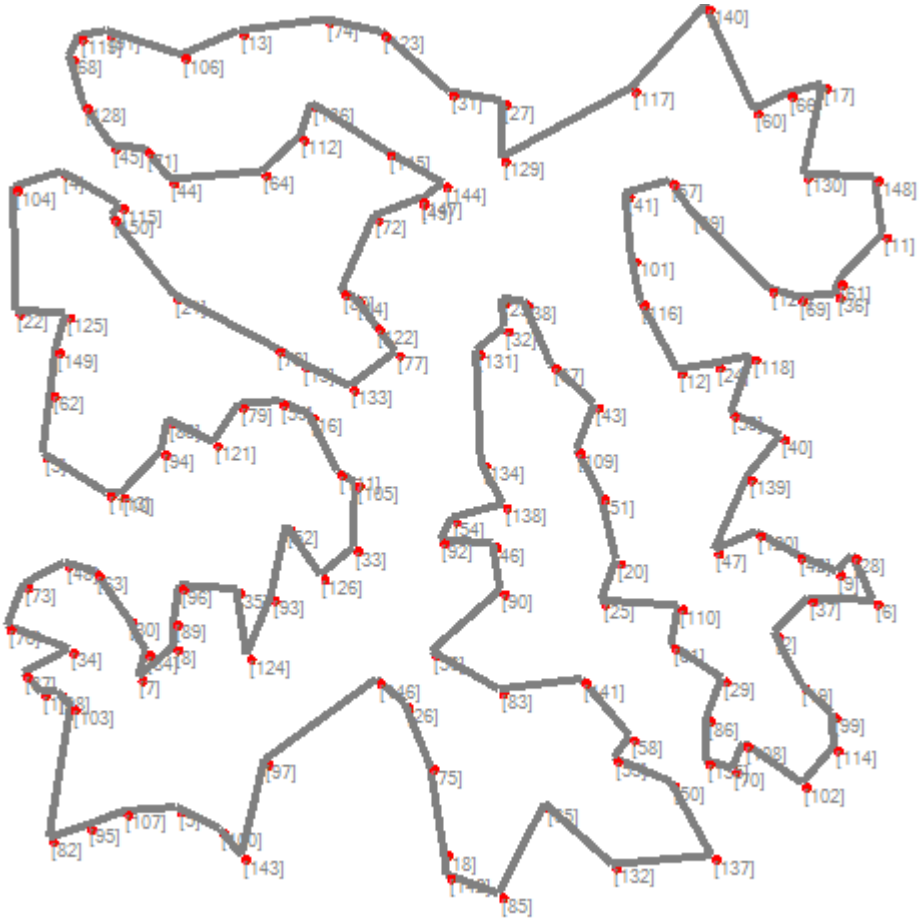

Figura 4.9. Representação do caminho da solução ótima da instância ch150 com a identificação de cada uma das cidades entre colchetes.

Como a instância ch150 possui *150* cidades, existem aproximadamente $3{,}81 \times 10^{260}$ caminhos possíveis. O caminho-solução descoberto pelo algoritmo para essa instância surgiu após *112109* passos de tempo em aproximadamente *39min57s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *99,27%* do seu peso total em relação ao peso total do caminho da solução ótima. O caminho-solução encontrado pelo algoritmo está representado na figura 4.10.

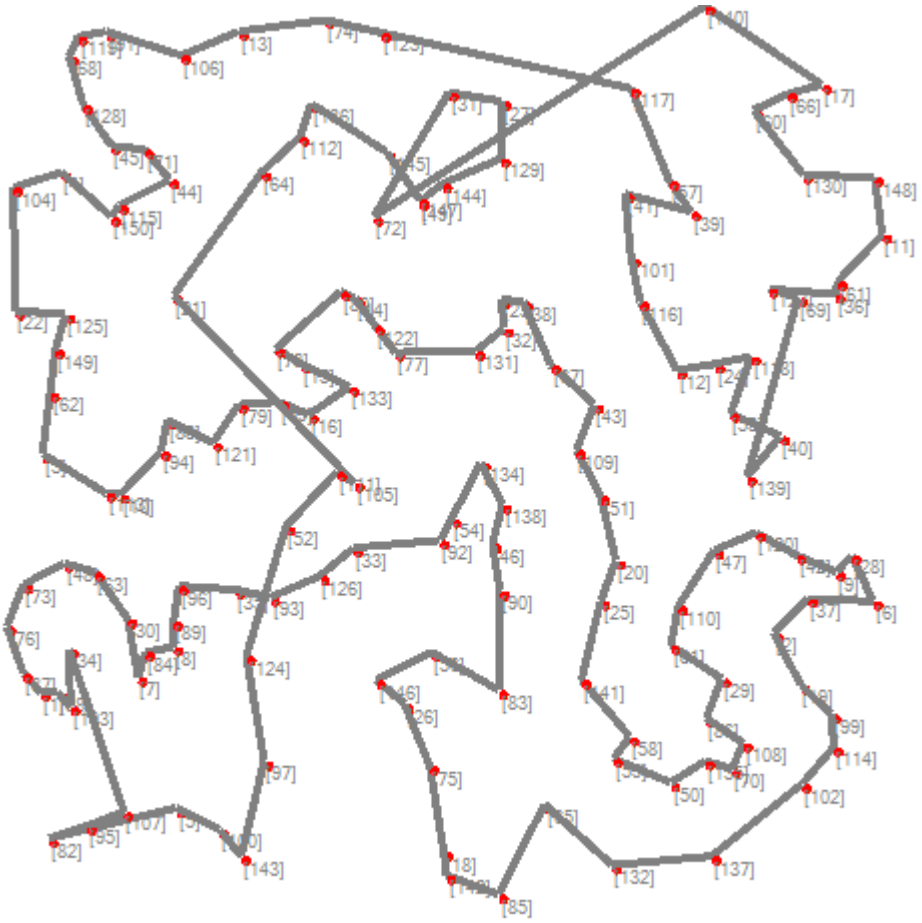

Figura 4.10. Representação do caminho-solução encontrado pelo algoritmo para a instância ch150 com a identificação de cada uma das cidades entre colchetes.

A instância eil101 sofreu uma pequena alteração para ser utilizada nos testes. Como as coordenadas de suas cidades possuem valor muito baixo ($X_{máximo}=67$ e $Y_{máximo}=77$) em relação às demais instâncias, todas as coordenadas foram multiplicadas por 10 para permitir uma melhor representação gráfica. Assim, é possível visualizar a instância eil101 através da figura 4.11, com sua matriz de dimensões transformadas para $L=680$ e $H=780$.

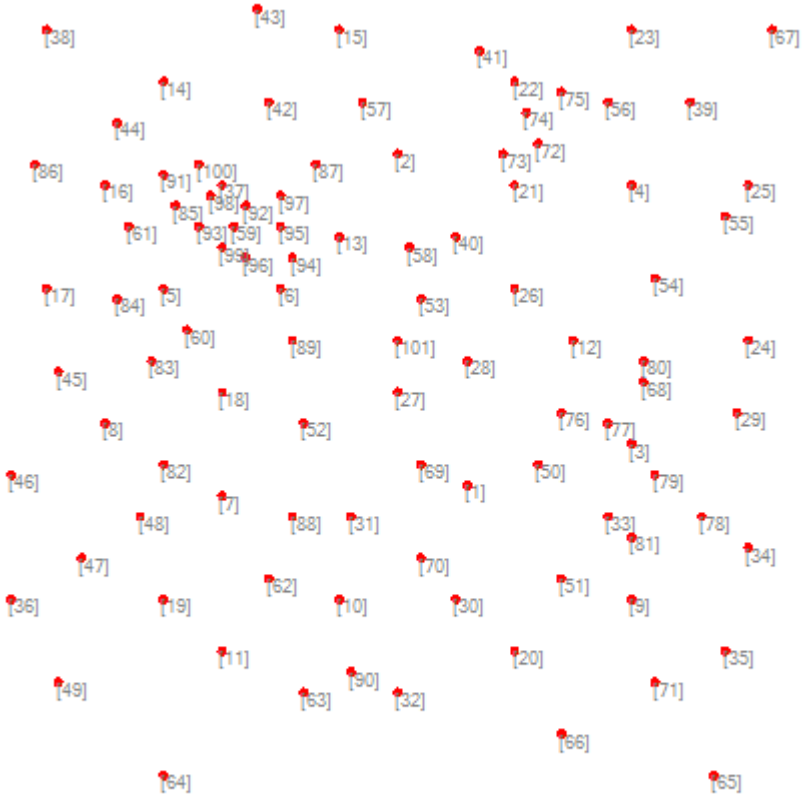

Figura 4.11. Representação das 101 cidades da instância eil101 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância eil101 tem peso total de aproximadamente *1,96*. O caminho da solução ótima pode ser visto na figura 4.12.

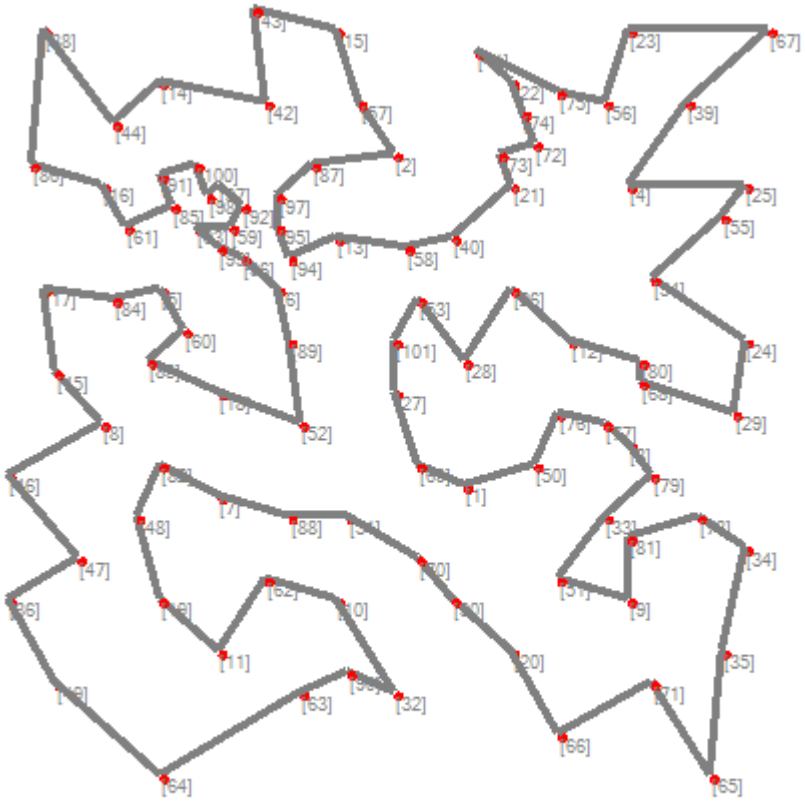

Figura 4.12. Representação do caminho da solução ótima da instância eil101 com a identificação de cada uma das cidades entre colchetes.

Como a instância eil101 possui *101* cidades, existem aproximadamente $9{,}33 \times 10^{157}$ caminhos possíveis. O caminho-solução apresentado pelo algoritmo para essa instância surgiu após *16571* passos de tempo em aproximadamente *5min31s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *99,00%*. Através da figura 4.13 é possível observar o caminho-solução encontrado pelo algoritmo.

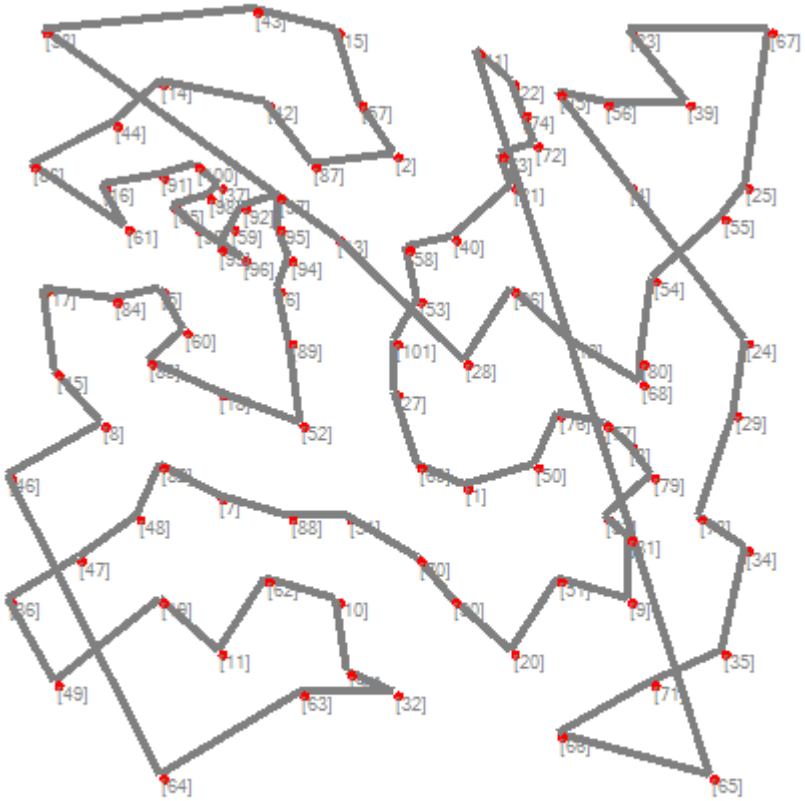

Figura 4.13. Representação do caminho-solução encontrado pelo algoritmo para a instância eil101 com a identificação de cada uma das cidades entre colchetes.

Assim como a instância eil101, a instância eil51 também foi adaptada para ser utilizada nos testes. Com as coordenadas de suas cidades apresentando também valor muito baixo ($X_{máximo}=63$ e $Y_{máximo}=69$) em relação às demais instâncias, todas as coordenadas foram multiplicadas por 10 para permitir uma representação gráfica mais adequada. A instância eil51 pode ser observada na figura 4.14, com sua matriz de dimensões transformadas para $L=640$ e $H=700$.

Figura 4.14. Representação das 51 cidades da instância eil51 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância eil51 possui peso total de aproximadamente *0,68*. O caminho da solução ótima está mostrado na figura 4.15.

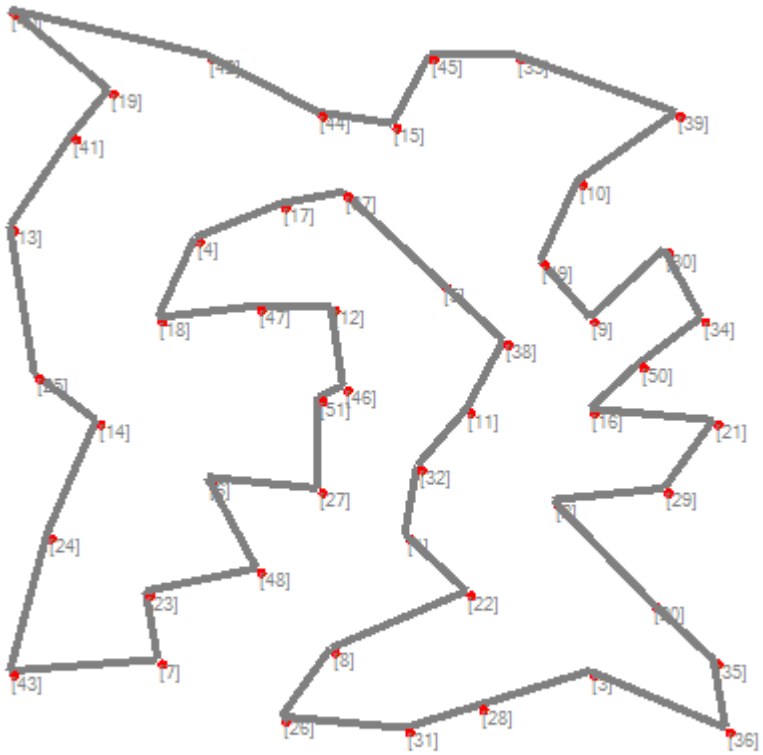

Figura 4.15. Representação do caminho da solução ótima da instância eil51 com a identificação de cada uma das cidades entre colchetes.

A instância eil51, ao apresentar *51* cidades, possibilita aproximadamente $3{,}04 \times 10^{64}$ caminhos. O caminho-solução gerado pelo algoritmo para essa instância surgiu após *231617* passos de tempo em aproximadamente *11min27s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *99,56%*. O caminho-solução encontrado pelo algoritmo é exibido na figura 4.16.

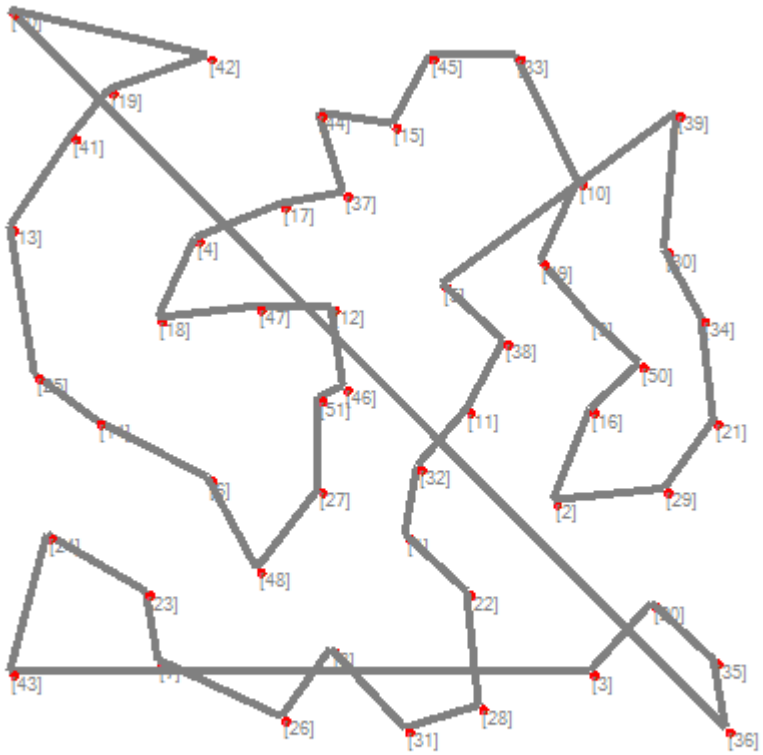

Figura 4.16. Representação do caminho-solução encontrado pelo algoritmo para a instância eil51 com a identificação de cada uma das cidades entre colchetes.

Como as instâncias anteriores eil101 e eil51, a instância eil76 também foi adaptada para ser utilizada nos testes, por conta de suas coordenadas das cidades apresentando também valor muito baixo ($X_{máximo}=70$ e $Y_{máximo}=76$) em relação às demais instâncias. Assim, todas as coordenadas foram multiplicadas por 10 para permitir uma representação gráfica mais adequada. A instância eil76 pode ser visualizada através da figura 4.17, com sua matriz de dimensões transformadas para $L=710$ e $H=770$.

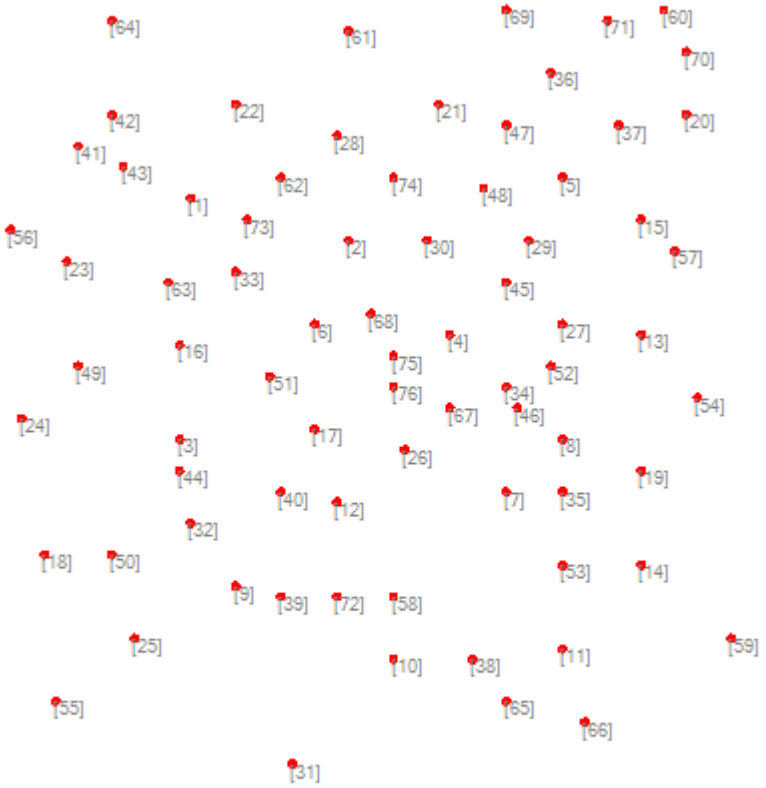

Figura 4.17. Representação das 76 cidades da instância eil76 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância eil76 apresenta peso total de aproximadamente *1,22*. O caminho da solução ótima é mostrado na figura 4.18.

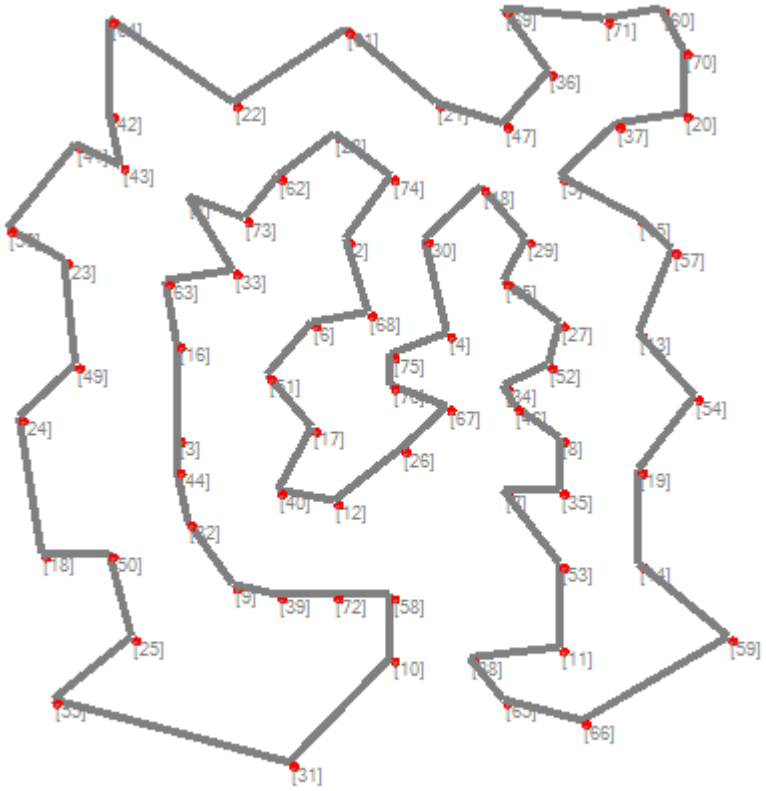

Figura 4.18. Representação do caminho da solução ótima da instância eil76 com a identificação de cada uma das cidades entre colchetes.

Como a instância eil76 apresenta *76* cidades, ela permite aproximadamente $2{,}48 \times 10^{109}$ caminhos. O caminho-solução gerado pelo algoritmo para essa instância surgiu após *13242* passos de tempo em aproximadamente *44s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *98,94%* do seu peso total em relação ao peso total do caminho da solução ótima. O caminho-solução encontrado pelo algoritmo é exibido na figura 4.19.

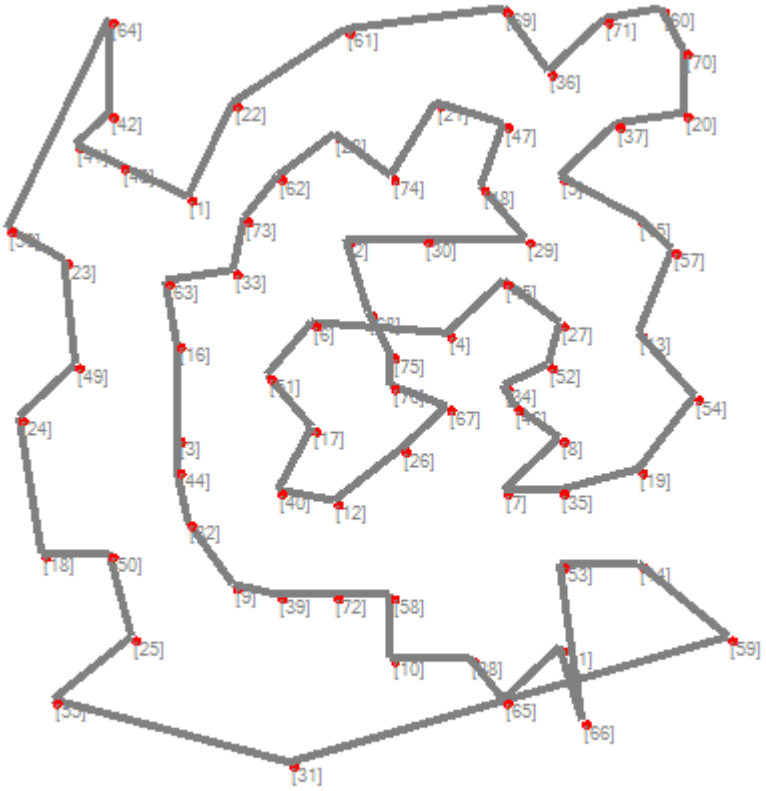

Figura 4.19. Representação do caminho-solução encontrado pelo algoritmo para a instância eil76 com a identificação de cada uma das cidades entre colchetes.

A instância kroC100 também foi utilizada nos testes. Com sua matriz de dimensões *L=3949* e *H=1967*, a instância kroC100 pode ser vista na figura 4.20.

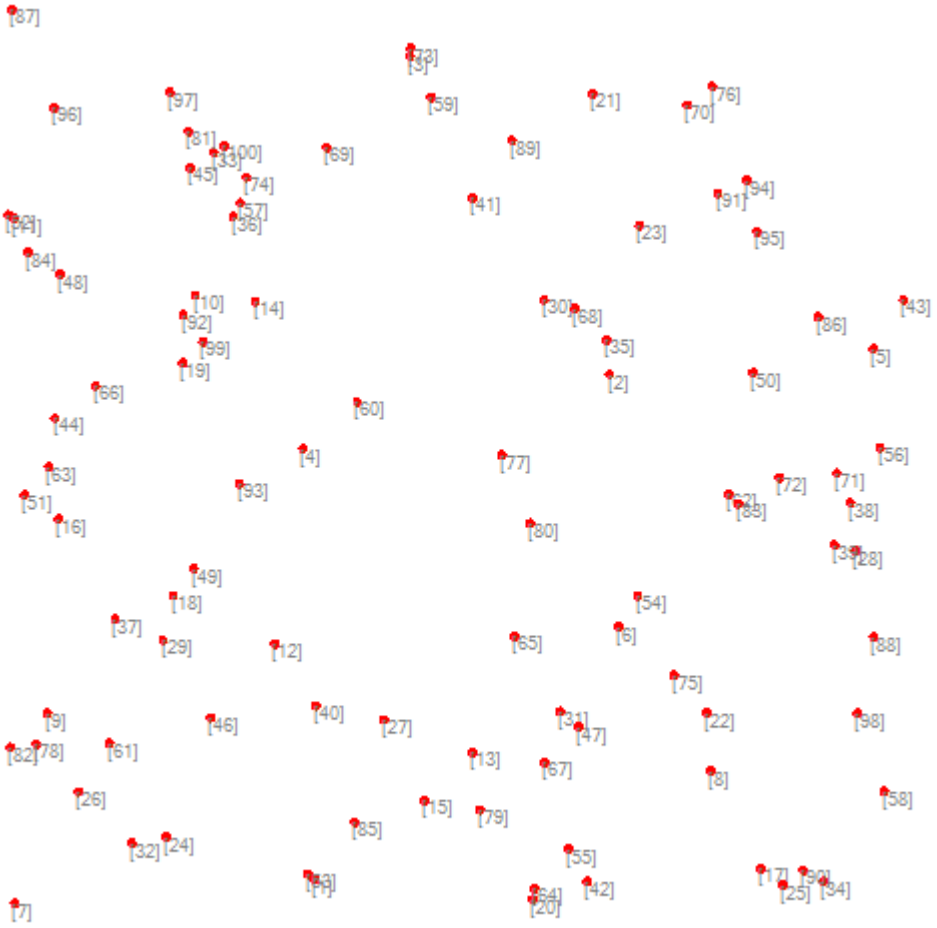

Figura 4.20. Representação das 100 cidades da instância kroC100 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância kroC100 apresenta peso total de aproximadamente *0,79*. O caminho da solução ótima pode ser visualizado na figura 4.21.

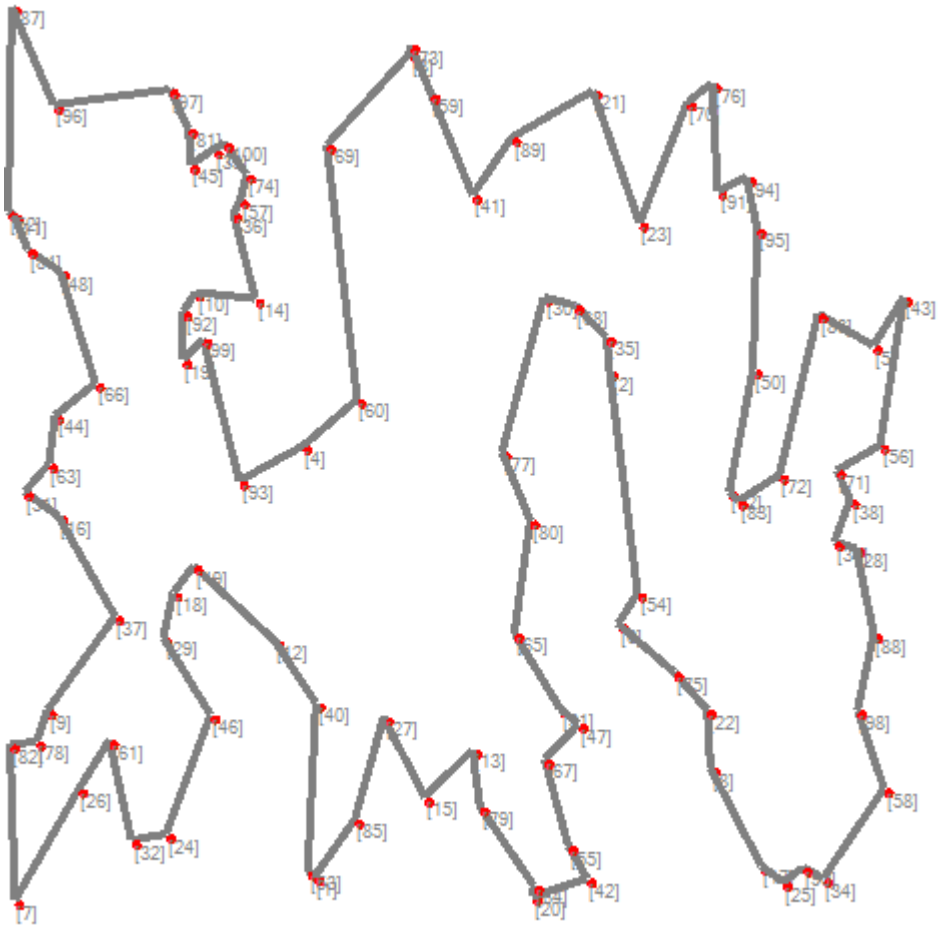

Figura 4.21. Representação do caminho da solução ótima da instância kroC100 com a identificação de cada uma das cidades entre colchetes.

A instância kroC100 possui *100* cidades, possibilitando a existência de $9{,}33 \times 10^{155}$ caminhos distintos. O caminho-solução apresentado pelo algoritmo para essa instância surgiu após *2931* passos de tempo em aproximadamente *16min19s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *99,31%*. A figura 4.22 representa o caminho-solução encontrado pelo algoritmo.

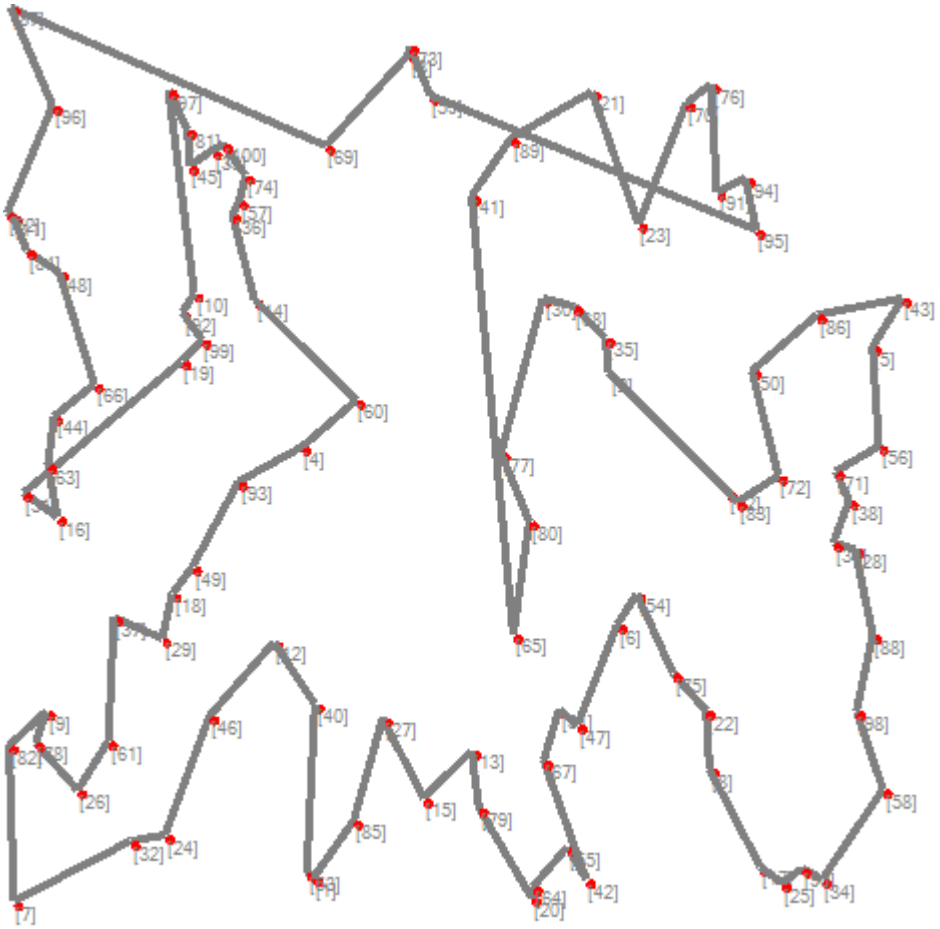

Figura 4.22. Representação do caminho-solução encontrado pelo algoritmo para a instância kroC100 com a identificação de cada uma das cidades entre colchetes.

Outra instância, a lin105, também foi utilizada nos testes. Com sua matriz de dimensões *L=3097* e *H=1286*, a instância lin105 está ilustrada na figura 4.23.

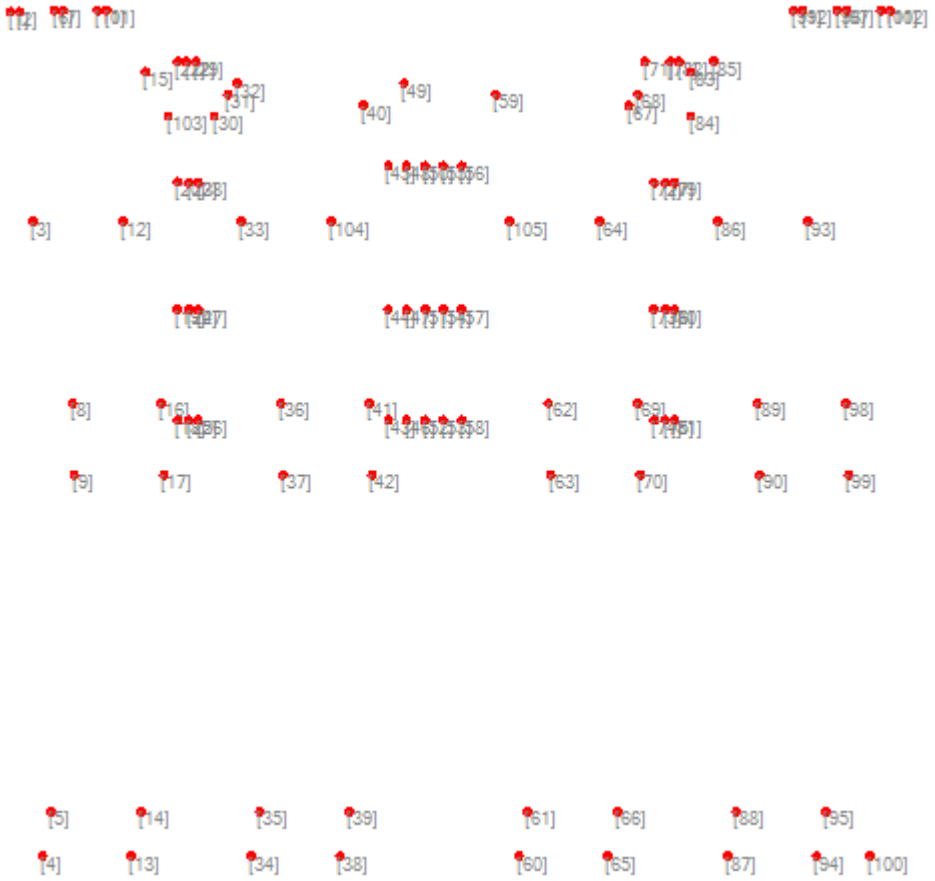

Figura 4.23. Representação das 105 cidades da instância lin105 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância lin105 apresenta peso total de aproximadamente *1,44*. O caminho da solução ótima está mostrado na figura 4.24.

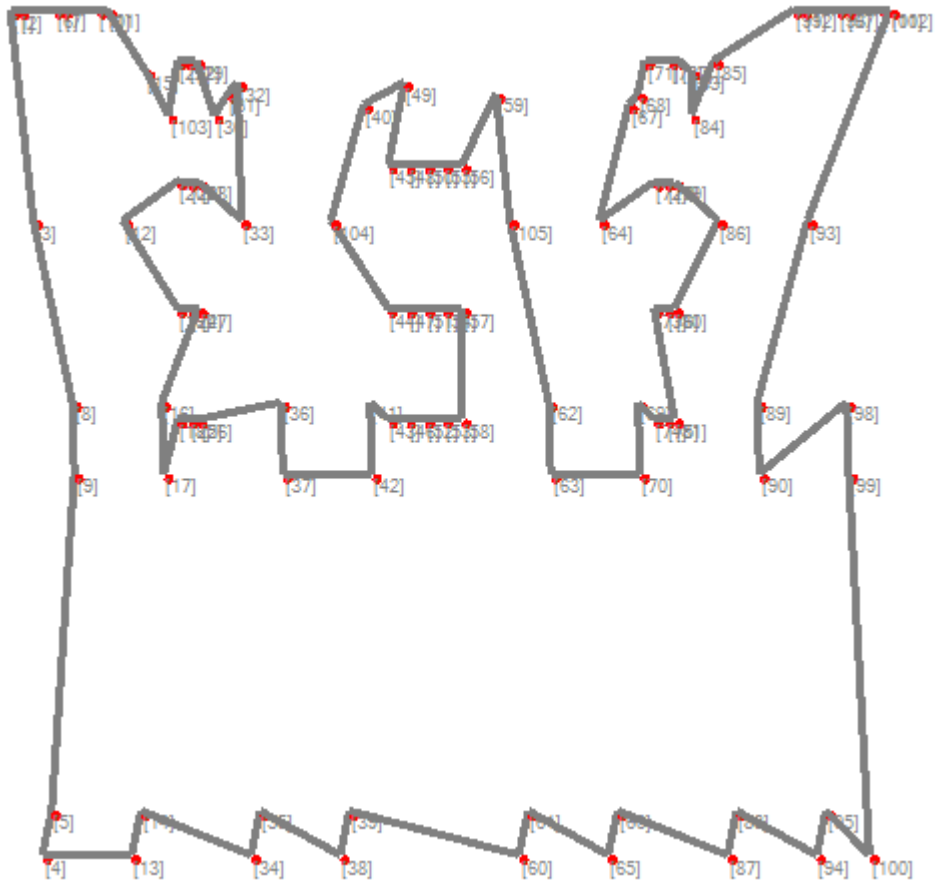

Figura 4.24. Representação do caminho da solução ótima da instância lin105 com a identificação de cada uma das cidades entre colchetes.

Como a instância lin105 possui *105* cidades, ela possibilita a existência de aproximadamente $1{,}03 \times 10^{166}$ caminhos. O caminho-solução encontrado pelo algoritmo para essa instância surgiu após *19344* passos de tempo em aproximadamente *5min45s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *99,27%*. A figura 4.25 mostra o caminho-solução encontrado pelo algoritmo.

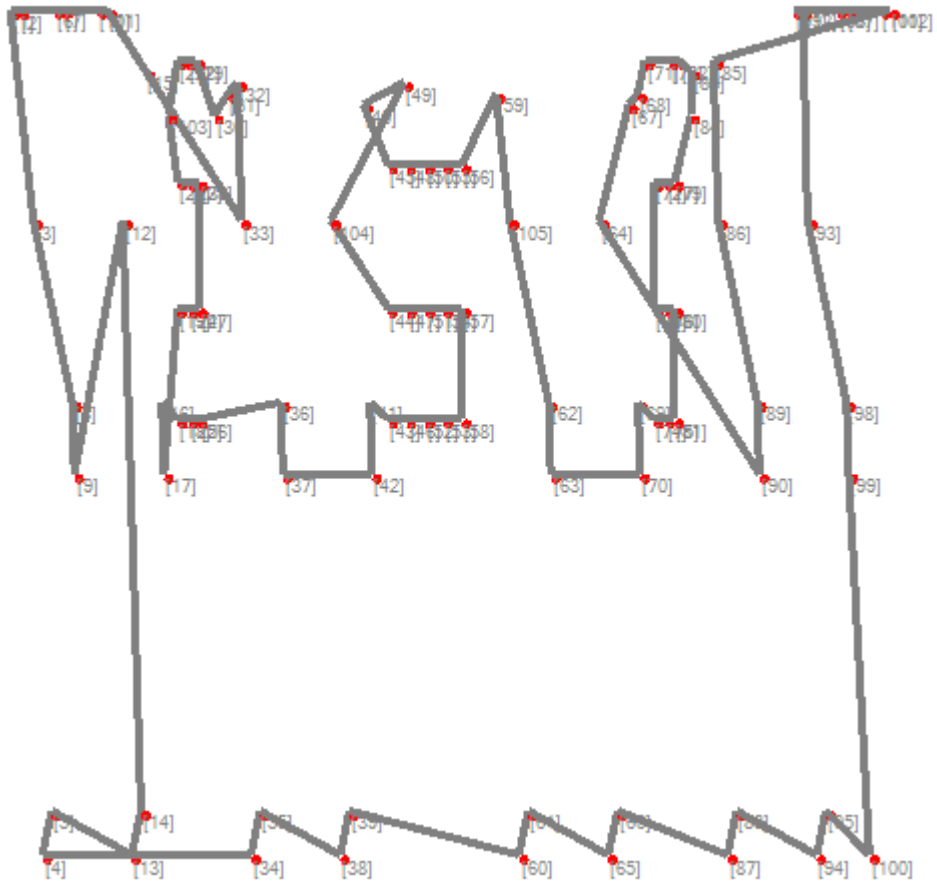

Figura 4.25. Representação do caminho-solução encontrado pelo algoritmo para a instância lin105 com a identificação de cada uma das cidades entre colchetes.

Na sequência, a instância pcb442 foi utilizada nos testes. Com sua matriz de dimensões *L=3010* e *H=3810* a instância pcb442 está mostrada na figura 4.26.

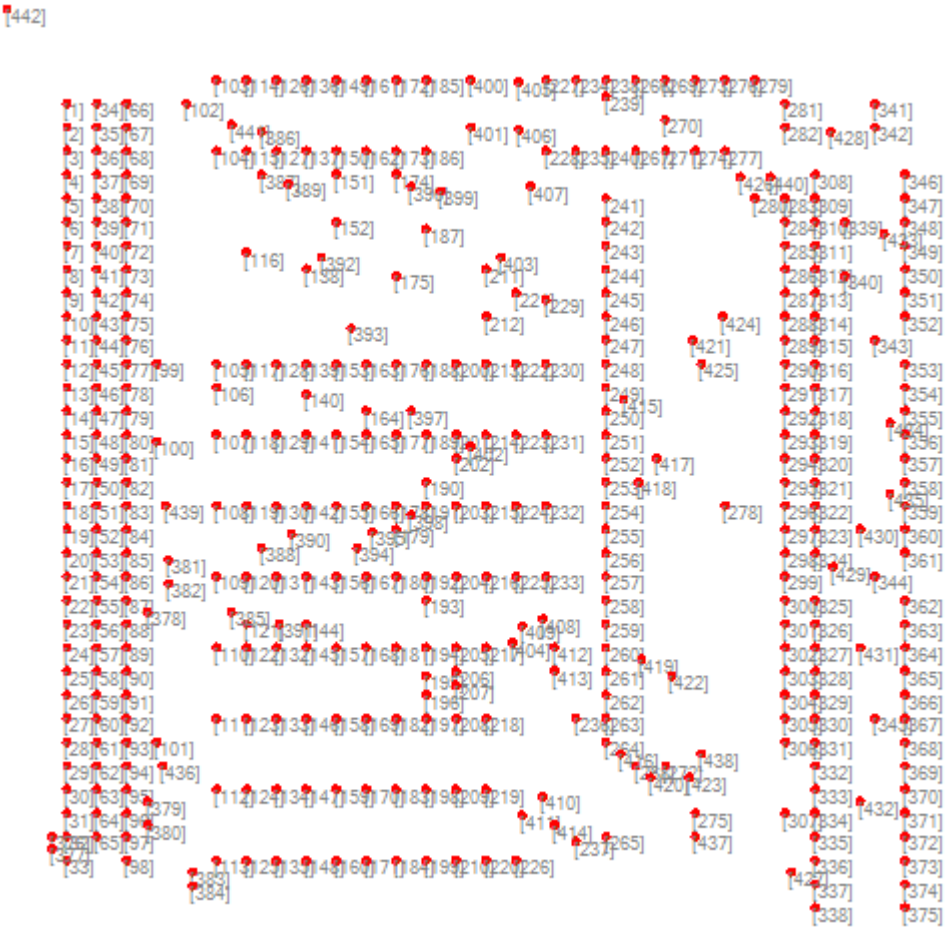

Figura 4.26. Representação das 442 cidades da instância pcb442 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância pcb442 tem peso total de aproximadamente *4,19*. O caminho da solução ótima pode ser visto na figura 4.27.

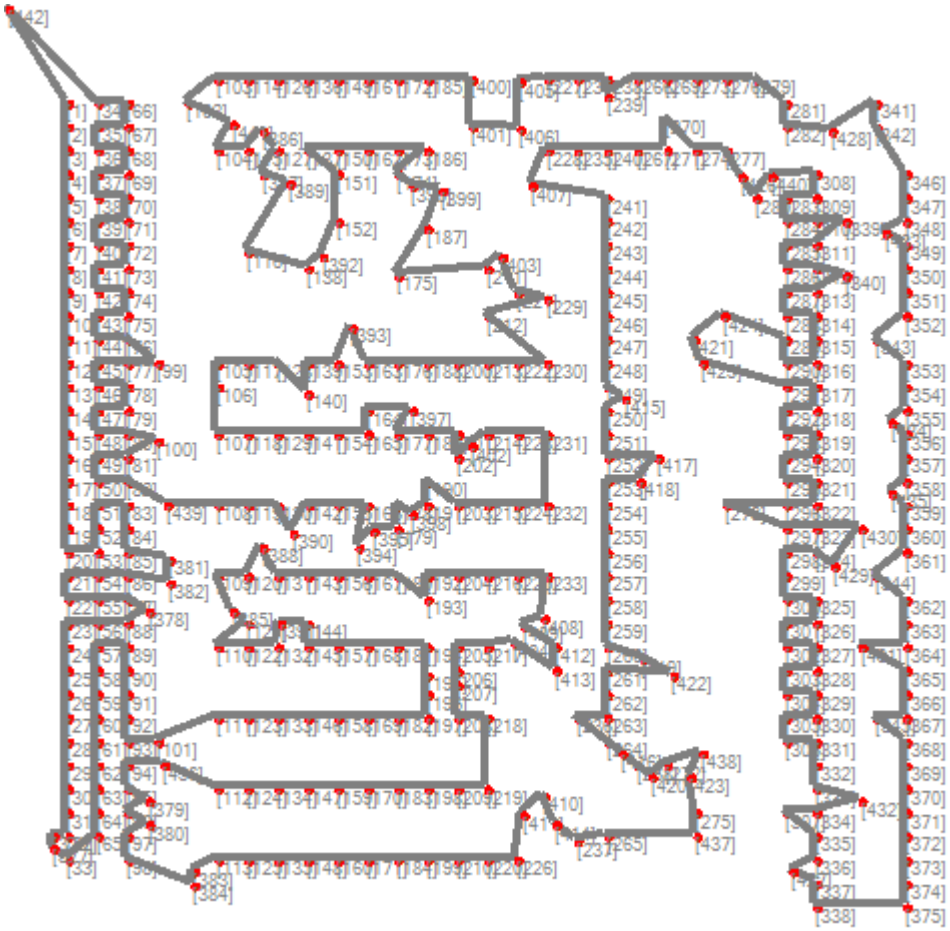

Figura 4.27. Representação do caminho da solução ótima da instância pcb442 com a identificação de cada uma das cidades entre colchetes.

A instância pcb442, ao apresentar *442* cidades, permite aproximadamente $2{,}48 \times 10^{976}$ caminhos possíveis. O caminho-solução achado pelo algoritmo para essa instância surgiu após *5637* passos de tempo em aproximadamente *7min46s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *98,02%*. O caminho-solução encontrado pelo algoritmo é mostrado pela figura 4.28.

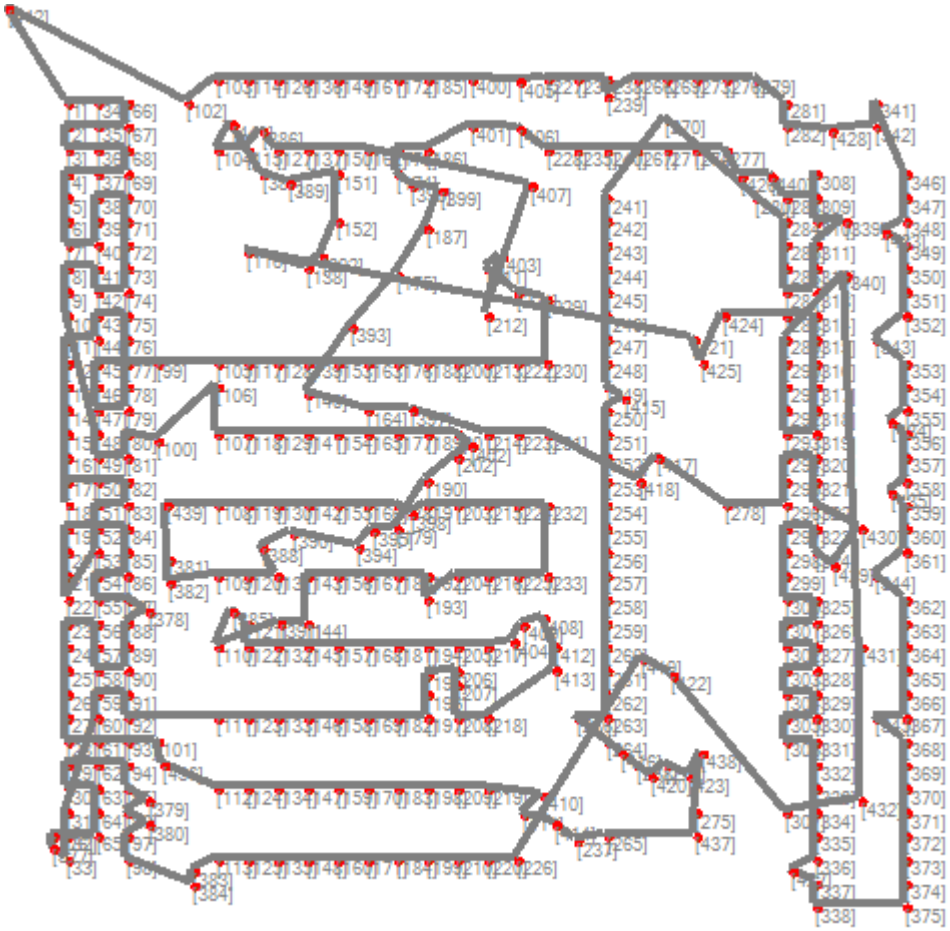

Figura 4.28. Representação do caminho-solução encontrado pelo algoritmo para a instância pcb442 com a identificação de cada uma das cidades entre colchetes.

Em seguida, a instância tsp225 foi utilizada nos testes. Sua matriz tem dimensões *L=635* e *H=367*. Essa instância pode ser vista na figura 4.29.

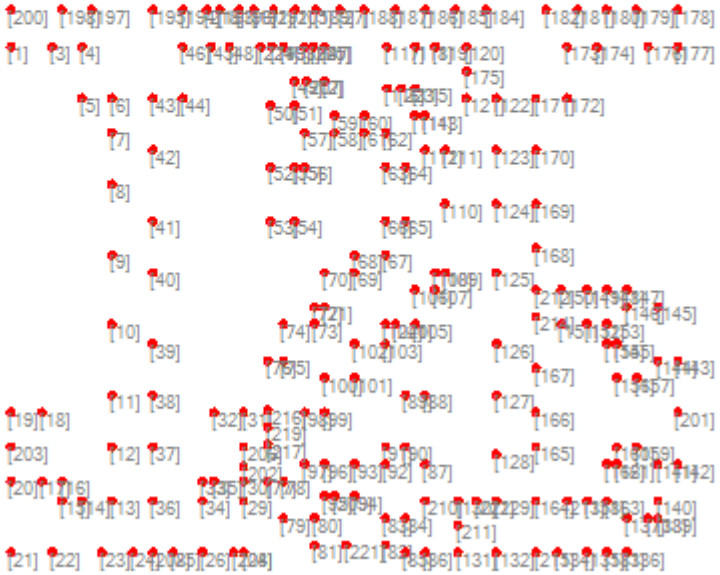

Figura 4.29. Representação das 225 cidades da instância tsp225 com a identificação de cada uma das cidades entre colchetes.

A solução ótima da instância tsp225 apresenta peso total de aproximadamente *15,80*. O caminho da solução ótima é destacado na figura 4.30.

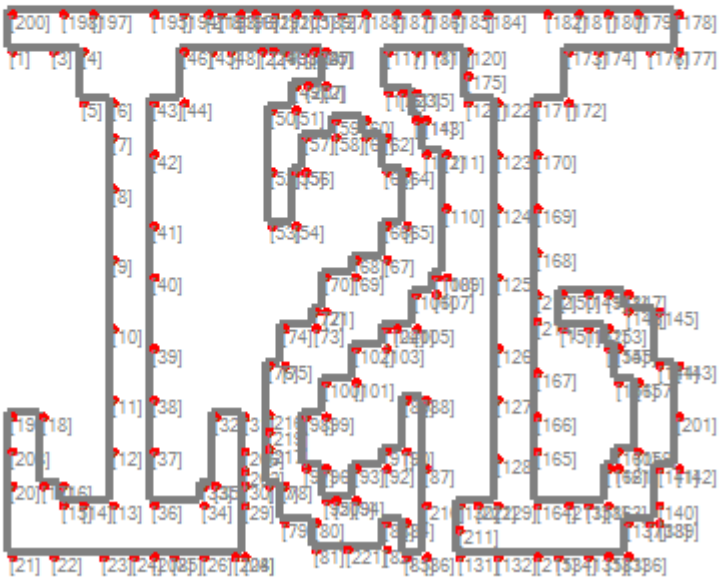

Figura 4.30. Representação do caminho da solução ótima da instância tsp225 com a identificação de cada uma das cidades entre colchetes.

A instância tsp225 apresenta *225* cidades, possibilitando assim aproximadamente $5{,}60 \times 10^{430}$ caminhos distintos. O caminho-solução apresentado pelo algoritmo para essa instância surgiu após *30810* passos de tempo em aproximadamente *1h11min42s* de tempo de execução. A qualidade do caminho-solução foi de aproximadamente *99,31%*. O caminho-solução encontrado pelo algoritmo foi modelado conforme a figura 4.31.

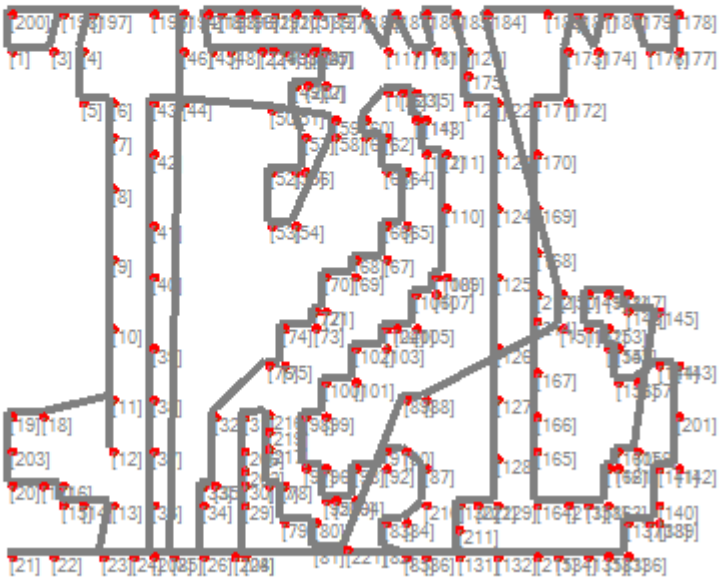

Figura 4.31. Representação do caminho-solução encontrado pelo algoritmo para a instância tsp225 com a identificação de cada uma das cidades entre colchetes.

A tabela 4.5 mostra as dimensões adotadas (quantidade de cidades, largura da matriz e altura da matriz) para cada instância avaliada.

Tabela 4.5. Dimensões de cada instância.

Dimensão	a280	berlin52	ch150	eil101	eil51	eil76	kroC100	lin105	pcb442	tsp225
Cidades	*N=280*	*N=52*	*N=150*	*N=101*	*N=51*	*N=76*	*N=100*	*N=105*	*N=442*	*N=225*
Largura	*L=298*	*L=1750*	*L=707*	*L=680*	*L=640*	*L=710*	*L=3949*	*L=3097*	*L=3010*	*L=635*
Altura	*H=179*	*H=1185*	*H=710*	*H=780*	*H=700*	*H=770*	*H=1967*	*H=1286*	*H=3810*	*H=367*

A tabela 4.6 apresenta os valores dos parâmetros utilizados pelo algoritmo v6 em todas as instâncias testadas. Esses valores foram escolhidos em função de testes preliminares realizados com a base berlin52 e usados nas demais instâncias. Não garantimos que eles são os ótimos para berlin52 e nem para as outras instâncias.

Tabela 4.6. Sumário dos valores de parâmetros utilizados em todas as instâncias.

Parâmetro	Valor
Fator de decremento	$\eta=0,00001$
Fator de incremento	$\mu=0,00002$
Limiar da função de ativação	$\lambda=1,0$
Passos no estado ativo	$\alpha=1$
Passos no estado refratário	$\beta=1$
Constante de saturação	$\varphi=1,0$

Os resultados encontrados para todas as instâncias testadas foram divididos em duas tabelas para melhor apresentação. Tais resultados são apresentados nas tabelas 4.7 e 4.8. Nessas tabelas, a ordem de grandeza mostra o valor aproximado de número de caminhos possíveis (que é *(N-1)!*); a qualidade indica a proximidade da melhor solução encontrada pelo algoritmo (mostrada, para cada instância, nas figuras anteriores) em relação à solução ótima; o tempo de simulação corresponde ao tempo total consumido pelas 100 simulações; o número de caminhos distintos indica a quantidade de caminhos distintos encontrados pelo algoritmo v6 dentre os 100 caminhos gerados nas 100 simulações.

Tabela 4.7. Sumário dos resultados encontrados nas instâncias a280, berlin52, ch150, eil101 e eil51.

	a280	berlin52	ch150	eil101	eil51
Ordem de grandeza	$5,99\times10^{562}$	$1,55\times10^{66}$	$3,81\times10^{260}$	$9,33\times10^{157}$	$3,04\times10^{64}$
Qualidade	*97,44%*	*99,71%*	*99,27%*	*99,00%*	*99,56%*
Tempo de simulação	*98:34:12*	*0:26:36*	*21:18:45*	*2:30:41*	*0:44:23*
Caminhos distintos	*72*	*19*	*31*	*34*	*18*

Tabela 4.8. Sumário dos resultados encontrados nas instâncias eil76, kroC100, lin105, pcb442 e tsp225.

	eil76	kroC100	lin105	pcb442	tsp225
Ordem de grandeza	$2,48\times10^{109}$	$9,33\times10^{155}$	$1,03\times10^{166}$	$2,48\times10^{976}$	$5,60\times10^{430}$
Qualidade	*98,94%*	*99,31%*	*99,27%*	*98,02%*	*99,31%*
Tempo de simulação	*1:50:37*	*0:35:59*	*2:16:11*	*57:03:38*	*35:39:21*
Caminhos distintos	*23*	*20*	*25*	*87*	*30*

5 CONCLUSÕES

Foram desenvolvidos algoritmos para solucionar o problema do caixeiro viajante, um problema considerado computacionalmente complexo. Tais algoritmos, baseados em observações relacionadas às modificações sinápticas que ocorrem em redes neurais biológicas, apresentaram um desempenho bastante satisfatório, comparando-se com os resultados dos trabalhos de Potvin (1996) e Dorigo e Gambardella (1997). Por exemplo, considerando o espaço de busca de uma instância como pcb442, em que há aproximadamente $2{,}48\times10^{976}$ caminhos possíveis, o algoritmo v6 conseguiu encontrar uma solução *98,02%* próxima da solução ótima. Para tsp225, com $5{,}60\times10^{430}$ possibilidades, v6 gerou um caminho *99,31%* próximo do ótimo. Em trabalhos futuros, pode-se continuar a evolução desse algoritmo, visando gerar versões capazes de aproximar os resultados ainda mais da solução ótima, bem como consumir menos tempo de processamento. A evolução do algoritmo pode também ser direcionada para permitir sua aplicabilidade a instâncias como pla85900 (com 89500 cidades), maior instância existente atualmente na TSPLIB (APPLEGATE et al., 2003; REINELT, 2010), ou outras instâncias de mesma ordem de grandeza.

Nos experimentos realizados para instâncias geradas aleatoriamente com *N=10*, avaliou-se o comportamento da versão v1 diante da alteração dos valores dos diversos parâmetros que caracterizam esse algoritmo. Um estudo similar pode ser realizado para as demais versões. Talvez existam valores de parâmetros que façam a versão v6 produzir resultados ainda melhores do que aqueles que foram encontrados. Em trabalhos futuros, tais valores podem ser encontrados da mesma forma que os experimentos realizados com a versão v1, permitindo a obtenção de melhores resultados da versão v6, até mesmo para instâncias analisadas da TSPLIB.

Os algoritmos desenvolvidos neste trabalho tiveram uma inspiração neural. Pode-se dizer que essa abordagem é promissora, pois ela leva à descoberta de soluções próximas da ótima com um tempo de processamento razoável (da ordem de horas em computador pessoal). Aqui, estudou-se apenas o problema do caixeiro viajante simétrico, mas, num trabalho futuro, o caso assimétrico pode ser tratado de modo similar.

BIBLIOGRAFIA

ABBOTT, L. F. e NELSON, S. B. (2000). Synaptic plasticity: taming the beast, *Nature Neuroscience Supplement* 3, 1178-1183.

APPLEGATE, D.; BIXBY, R.; CHV´ATAL, V. e COOK, W. (2003). Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems, *Mathematical Programming* 97, 91-153.

BAGCHI, T. P.; GUPTA, J. N. D. e SRISKANDARAJAH, C. (2006). A review of TSP based approaches for flowshop scheduling, *European Journal of Operational Research* 169, 816-854.

BEAR, M. F.; CONNORS, B. W. e PARADISO, M. A. (2002). *Neurociências*, 2 ed., Artmed Editora.

BEAR, M.F. (2003). Bidirectional synaptic plasticity: from theory to reality, *Philosophical Transactions of Royal Society B (London)* 358, 649-655.

BUONOMANO, D. V. e MERZENICH, M. M. (1998). Cortical plasticity: from synapses to maps, *Annual Review of Neuroscience* 21, 149-186.

CHAO, T. C. e CHEN, C. M. (2005). Learning-induced synchronization and plasticity of a developing neural network, *Journal of Computational Neuroscience* 19, 311-324.

COOPER, S. J. (2005). Donald O. Hebb's synapse and learning rule: a history and commentary, *Neuroscience & Biobehavioral Reviews* 28, 851-874.

DORIGO, M.; MANIEZZO, V. e CORLONI, A. (1996). The ant system: optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics B* 26, 29-41.

DORIGO, M.; GAMBARDELLA, L. M. (1997). Ant colonies for the travelling salesman problem, *Biosystems* 43, 73-81.

FIORI, S. (2005). Nonlinear complex-valued extensions of Hebbian learning: an essay, *Neural Computation* 17, 779-838.

GERSTNER, W. e KISTLER, W. M. (2002). Mathematical formulations of hebbian learning, *Biological Cybernetics* 87, 404-415.

HAYKIN, S. (1998). *Neural Networks: A Comprehensive Foundation*, 2. ed, Prentice Hall.

KEMPTER, R.; GERSTNER, W. e LEO VAN HEMMEN, J. (1999). Hebbian learning and spiking neurons. *Physical Review E* 59, 4498-4514.

KOCH, C. (1999). *Biophysics of Computation*, 1. ed., Oxford University Press.

LIN, S. e KERNIGHAN, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem, *Operations Research* 21, 498-516.

MORRISON, A.; DIESMANN, M. e GERSTNER, W. (2008). Phenomenological models of synaptic plasticity based on spike timing, *Biological Cybernetics* 98, 459-478.

ONCANA, T.; ALTINELB, I. K. e LAPORTEC, G. (2009). A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers and Operations Research* 36, 637-654.

POTVIN, J. Y. (1996). Genetic algorithms for the traveling salesman problem, *Annals of Operations Research* 63, 339-370.

REINELT, G. (2010). TSPLIB 95. Traveling Salesman Problem Library (TSPLIB). Disponível em: <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/DOC.PS>. Acessado em 9 de junho de 2010.

ROBERTS, P. D. (1999). Computational consequences of temporally asymmetric learning rules: I. differential Hebbian learning, *Journal of Computational Neuroscience* 7, 235-246.

SCHRIJVER, A. (2005). On the history of combinatorial optimization (till 1960), In: Handbook of Discrete Optimization (K. AARDAL, G. L. NEMHAUSER, R. WEISMANTEL, eds.), Elsevier, 1-68.

SIRI, B.; QUOY, M.; DELORD, B.; CESSAC, B. e BERRY, H. (2007). Effects of Hebbian learning on the dynamics and structure of random networks with inhibitory and excitatory neurons, *Journal of Physiology (Paris)* 101, 136-148.

SONG, S. e ABBOTT, L. F. (2001). Cortical development and remapping through spike timing-dependent plasticity, *Neuron* 32, 339-350.

SONG, S.; MILLER, K. D. e ABBOTT, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity, *Nature Neuroscience* 3, 919-926.

SPORNS, O. (2006). Small-world connectivity, motif composition, and complexity of fractal neuronal connections, *Biosystems* 85, 55-64.

SPORNS, O. e ZWI, J. D. (2004). The small world of the cerebral cortex, *NeuroInformatics* 2, 145-162.

APÊNDICE A

Pseudocódigo de um passo de tempo do algoritmo v0.

```
01:       foreach(Neuron in Neurons)
02:       {
03:         Neuron.UpdateSynapses();
04:         Neuron.UpdateNeuron();
05:       }
06:
07:       InitialNeuron = Neurons[0];
08:
09:       Path = SearchPathGreedy(InitialNeuron);
10:
11:       SortedNeurons = Neurons.SortBySynapsesCount();
12:
13:       foreach(Neuron1 in SortedNeurons)
14:       {
15:         foreach(Neuron2 in SortedNeurons)
16:         {
17:           Synapse = Neuron1.GetSynapseTo(Neuron2);
18:           if(Synapse.Weight < 0.0 and not Path.Contains(Synapse))
19:           {
20:             Synapse.Destroy();
21:           }
22:         }
23:       }
```

As figuras A1 a A3 ajudam a explicar a lógica do algoritmo v0 durante sua execução.

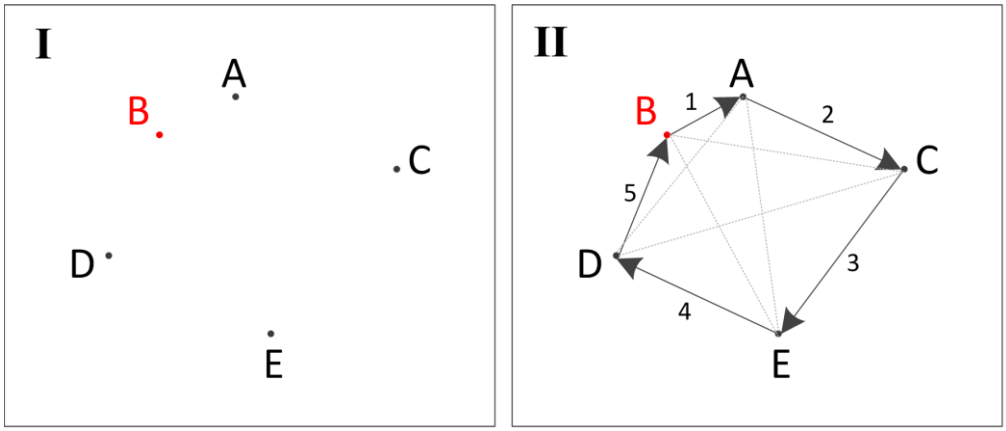

Figura A.1. O quadro I ilustra a linha 07 do pseudocódigo em que a cidade inicial (B) é sempre a mesma ao longo do tempo (a mais próxima do canto esquerdo superior). A linha 09 é representada pelo quadro II. Partindo de B, um caminho guloso é montado; ou seja, a cidade mais próxima ainda não visitada é escolhida como a seguinte, até passar por todas elas. As setas indicam a ordem com que o caminho é formado (números de 1 a 5). As demais ligações existentes naquele instante são representadas pelas linhas pontilhadas.

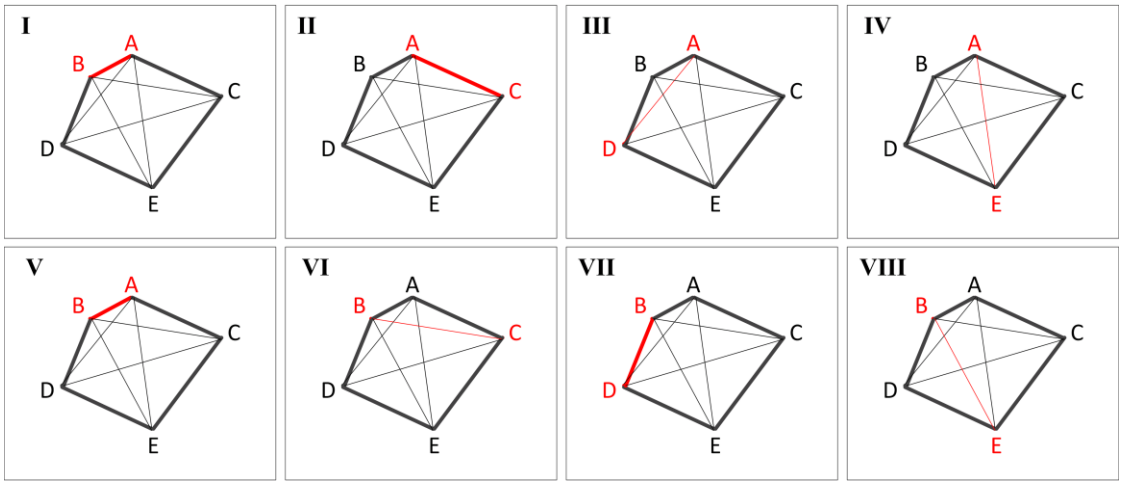

Figura A.2. Após a criação de uma lista ordenada de todas as cidades por ordem decrescente de número de ligações (linha 11 do pseudocódigo), algumas das interações geradas pelas linhas 13 a 23 são mostradas pelos quadros I a VIII. Cada par de cidades tem sua ligação avaliada. Os quadros I a IV mostram a relação de A com as demais cidades. Os quadros V a VIII mostram a relação de B com as demais cidades. As interações seguem até que todas as ligações tenham sido avalidas.

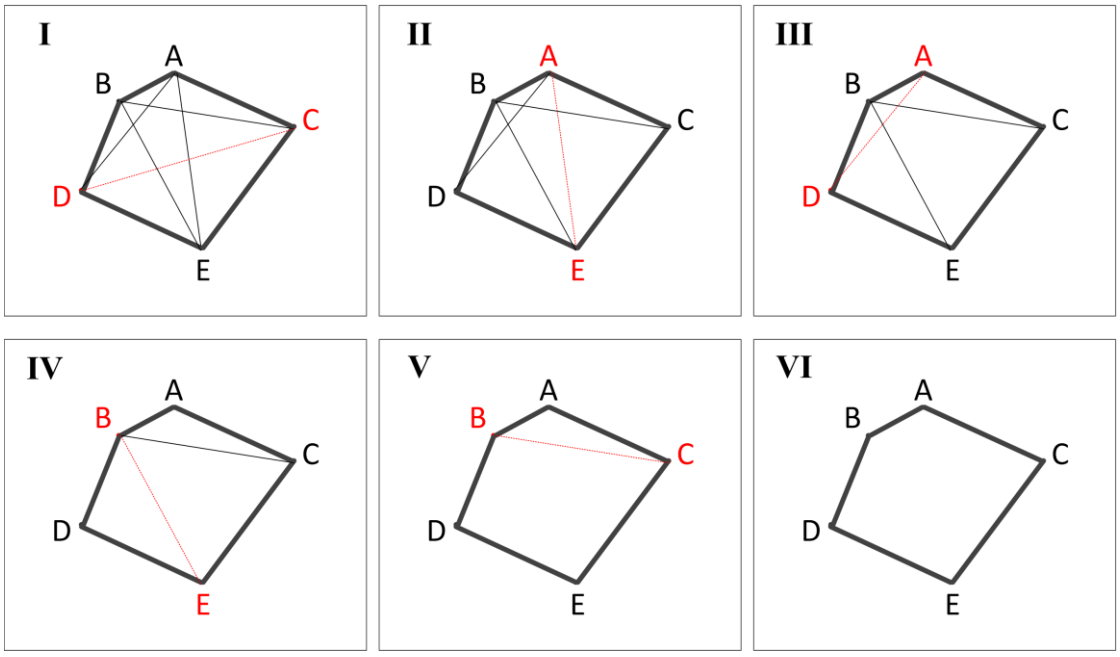

Figura A.3. Conforme o tempo passa, determinadas avaliações da ligação existente entre um par de cidades incorre na situação representada pelas linhas 18 a 21 do pseudocódigo. O quadro I ilustra a ligação entre C e D sendo avaliada. Ao apresentar peso menor ou igual a zero e não pertencer ao caminho daquele instante, a ligação é então eliminada. O quadro II mostra a ligação entre A e E sendo avaliada, já com a ausência da ligação eliminada anteriormente. O quadro III destaca a ligação entre A e D. Os demais quadros ilustram os passos de tempo seguintes. O quadro VI representa o caminho resultante após a eliminação de ligações ao longo do tempo. Esse caminho é então considerado o caminho-solução.

Pseudocódigo de um passo de tempo do algoritmo v1.

```
01:       InitialNeuron = Neurons.GetRandomNeuron();
02:
03:       Path = SearchPathGreedy(InitialNeuron);
04:
05:       foreach(Neuron in Neurons)
06:       {
07:         Neuron.ProcessInput(Neuron.NextInput);
08:       }
09:
10:       foreach(Neuron in Neurons)
11:       {
12:         Neuron.UpdateSynapses();
13:         Neuron.UpdateNeuron();
14:       }
15:
16:       SortedNeurons = Neurons.SortBySynapsesCount();
17:
18:       foreach(Neuron1 in SortedNeurons)
19:       {
20:         foreach(Neuron2 in SortedNeurons)
21:         {
22:           Synapse = Neuron1.GetSynapseTo(Neuron2);
23:           if(Synapse.Weight < 0.0 and not Path.Contains(Synapse))
24:           {
25:             Neuron1.NextInput = FIRE;
26:             Neuron2.NextInput = FIRE;
27:             Synapse.Destroy();
28:           }
29:         }
30:       }
```

As diferenças entre o algoritmo v1 e o algoritmo anterior v0 estão destacadas em amarelo nas linhas 01, 25 e 26. As figuras A.4 a A.6 procuram ilustrar essas diferenças.

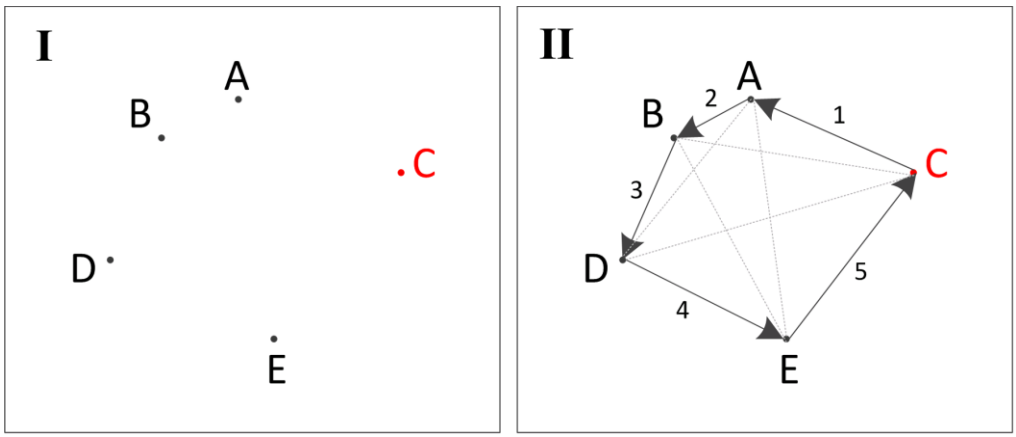

Figura A.4. O quadro I ilustra a linha 01 do pseudocódigo em que a cidade inicial é escolhida aleatoriamente (C, no caso). Em seguida, partindo de C, um caminho guloso é montado. O quadro II ilustra o caminho seguido a partir da cidade inicial. As setas indicam o caminho na ordem que foi construído (números de 1 a 5). As demais ligações existentes naquele instante são representadas pelas linhas pontilhadas.

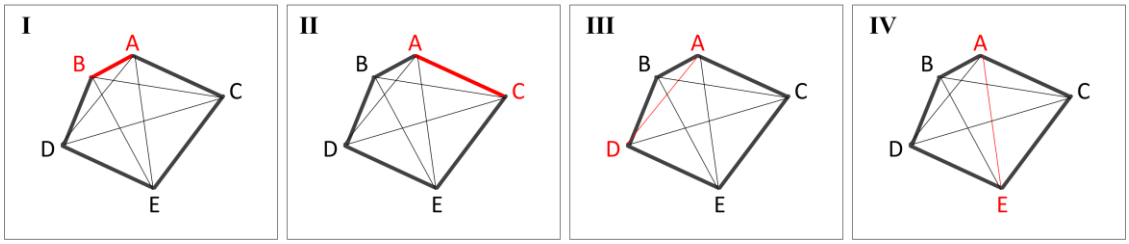

Figura A.5. Cria-se uma lista ordenada de todas as cidades por ordem decrescente de número de ligações (linha 16 do pseudocódigo). Os quadros I a IV mostram algumas das interações geradas pelas linhas 18 a 30 do pseudocódigo. Cada par de cidades tem sua ligação avaliada. Os quadros I a IV representam a avaliação da ligação de A com as demais cidades. Assim como no algoritmo v0, as interações seguem até que todas as ligações tenham sido avalidas.

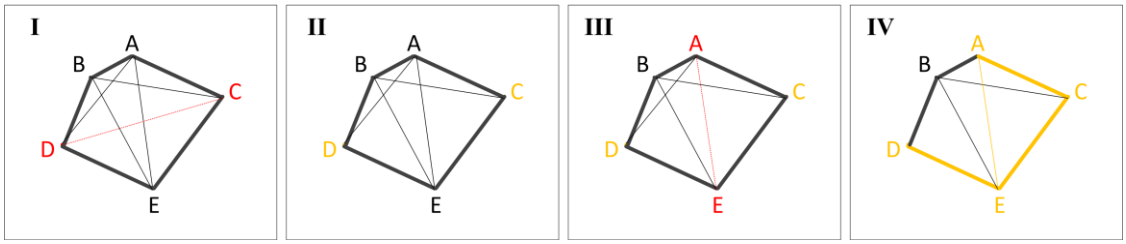

Figura A.6. Com o passar do tempo, conforme definido pelas linhas 23 a 28 do pseudocódigo, determinadas ligações são eliminadas ao apresentar peso menor ou igual a zero e não pertencer ao caminho daquele instante. O quadro I ilustra a ligação entre C e D sendo avaliada nessa situação. Ao ser eliminada, o par de cidades que essa ligação conectava sofre um estímulo. O quadro II ilustra C e D no estado ativo após serem estimuladas (em amarelo) e já com a ausência da ligação eliminada anteriormente. A ligação entre A e E (em vermelho), ao se tornar negativa ou nula, também é eliminada, fazendo com que A e E também disparem (em amarelo). Assim, neurônios que ficam no estado ativo simultaneamente fortalecem as ligações existentes entre eles. O quadro IV representa A, C, D e E no estado ativo e, consequentemente, há o fortalecimento das respectivas ligações (em amarelo). O processo todo segue até que reste apenas um único caminho possível, considerado então o caminho-solução.

Pseudocódigo de um passo de tempo do algoritmo v2.

```
01:       InitialNeuron = Neurons.GetRandomNeuron();
02:
03:       Path = SearchPathGreedy(InitialNeuron);
04:
05:       foreach(Synapse in Synapses)
06:       {
07:         if(Path.Contains(Synapse))
08:         {
09:           Synapse.Type = EXCITATORY;
10:         }
11:         else
12:         {
13:           Synapse.Type = INHIBITORY;
14:         }
15:       }
16:
17:       foreach(Neuron in Neurons)
18:       {
19:         Neuron.ProcessInput(FIRE);
20:       }
21:
22:       foreach(Neuron in Neurons)
23:       {
24:         Neuron.UpdateSynapses();
25:         Neuron.UpdateNeuron();
26:
27:         foreach(Synapse in Synapses)
28:         {
29:           if(Synapse.Weight < 0.0 and not Path.Contains(Synapse))
30:           {
31:             Synapse.Destroy();
32:           }
33:         }
34:       }
```

As diferenças entre os algoritmos v2 e v1 estão realçadas em amarelo nas linhas 05 a 20 do pseudocódigo. A figura A.7 mostra essas diferenças.

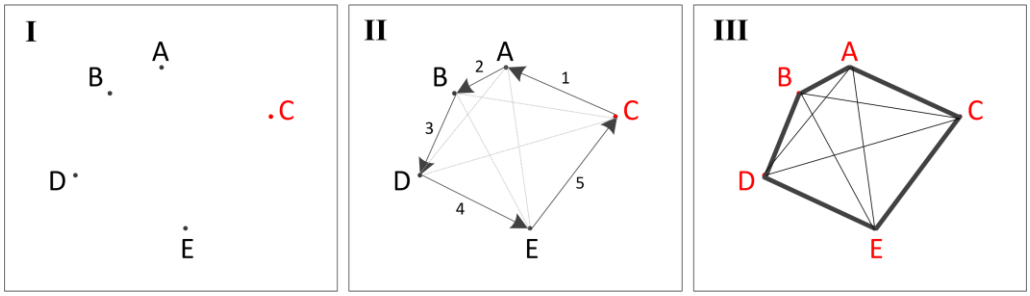

Figura A.7. Como na versão v1, o quadro I ilustra a linha 01 do pseudocódigo, em que a cidade inicial é escolhida aleatoriamente (C). Em seguida, partindo de C um caminho guloso é montado. A principal diferença em relação a v1 consiste nas linhas 05 a 20 do pseudocódigo e pode ser observada no quadro III. Nelas, as ligações que formam o caminho guloso são definidas como sinapses excitatórias (ligações em negrito) e as demais como sinapses inibitórias (ligações num tom de cinza). Todos os neurônios são então estimulados para dispararem (em vermelho). Da mesma forma que nos algoritmos anteriores, ligações negativas ou nulas são eliminadas. O processo segue também até restar um único caminho, que é considerado o caminho-solução.

Pseudocódigo de um passo de tempo do algoritmo v3.

```
01:       do
02:       {
03:         InitialNeuron = Neurons.GetRandomNeuron();
04:       } while(UsedNeurons.Contains(InitialNeuron));
05:
06:       UsedNeurons.Add(InitialNeuron);
07:
08:       if(UsedNeurons.Count >= Neurons.Count)
09:       {
10:         UsedNeurons.Clear();
11:       }
12:
13:       Path = SearchPathGreedy(InitialNeuron);
14:
15:       foreach(Synapse in Synapses)
16:       {
17:         if(Path.Contains(Synapse))
18:         {
19:           Synapse.Type = EXCITATORY;
20:         }
21:         else
22:         {
23:           Synapse.Type = INHIBITORY;
24:         }
25:       }
26:
27:       foreach(Neuron in Neurons)
28:       {
29:         Neuron.ProcessInput(FIRE);
30:       }
31:
32:       foreach(Neuron in Neurons)
33:       {
34:         Neuron.UpdateSynapses();
35:         Neuron.UpdateNeuron();
36:
37:         foreach(Synapse in Synapses)
38:         {
39:           if(Synapse.Weight < 0.0 and not Path.Contains(Synapse))
40:           {
41:             Synapse.Destroy();
42:           }
43:         }
44:       }
```

As diferenças entre v3 e v2 estão destacadas em amarelo nas linhas 01 a 11 do pseudocódigo acima. A figura A.8 ilustra essas diferenças.

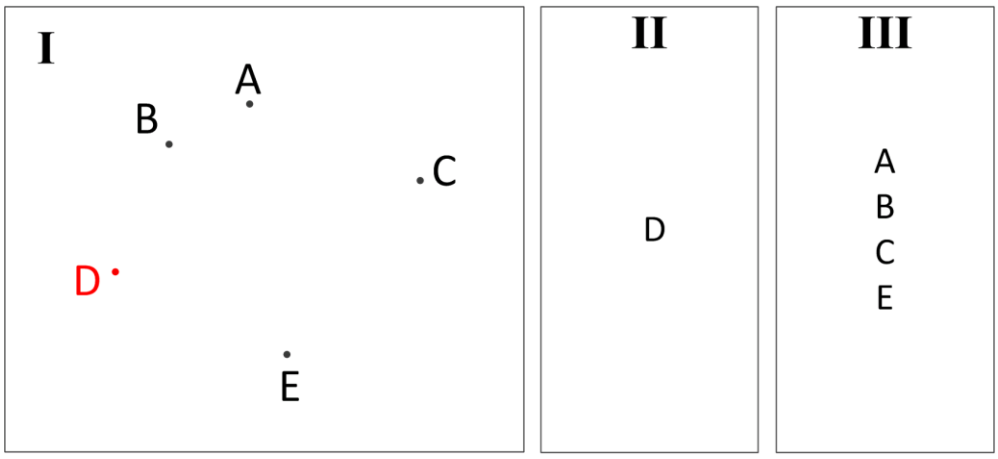

Figura A.8. Os quadros ilustram a diferença de comportamento em relação ao algoritmo anterior, definidos pelas linhas 01 a 11 do pseudocódigo. Considerando todas as cidades ainda não escolhidas como inicial, o quadro I mostra a escolha aleatória de D como sendo a cidade inicial. Essa cidade é então colocada em um conjunto de cidades escolhidas como inicial, conforme mostra o quadro II. O conjunto de cidades ainda não escolhidas como inicial é mostrado no quadro III. Em seguida, assim como no algoritmo anterior, partindo de D um caminho guloso é montado. As ligações que formam o caminho guloso são então definidas como sinapses excitatórias e as demais como inibitórias. Também, todos os neurônios são estimulados para dispararem e, da mesma forma que nos algoritmos anteriores, ligações negativas ou nulas são eliminadas. O processo segue também até restar um único caminho, chamado de caminho-solução. Com a repetição do processo de seleção da cidade inicial ao longo do tempo, todas são escolhidas como inicial por uma vez. Conforme as linhas 08 a 11 do pseudocódigo estabelecem, quando todas cidades já foram escolhidas, o conjunto mostrado pelo quadro II (escolhidas) é então esvaziado.

Pseudocódigo de um passo de tempo do algoritmo v4.

```
01:       do
02:       {
03:         InitialNeuron = Neurons.GetRandomNeuron();
04:       } while(UsedNeurons.Contains(InitialNeuron));
05:
06:       UsedNeurons.Add(InitialNeuron);
07:
08:       if(UsedNeurons.Count >= Neurons.Count)
09:       {
10:         UsedNeurons.Clear();
11:       }
12:
13:       Path = SearchPathGreedy(InitialNeuron);
14:
15:       if(Path.Weight > BestPath.Weight)
16:       {
17:         BestPath = Path;
18:       }
19:       else
20:       {
21:         Path = BestPath;
22:       }
23:
24:       foreach(Synapse in Synapses)
25:       {
26:         if(Path.Contains(Synapse))
27:         {
28:           Synapse.Type = EXCITATORY;
29:         }
30:         else
31:         {
32:           Synapse.Type = INHIBITORY;
33:         }
34:       }
35:
36:       foreach(Neuron in Neurons)
37:       {
38:         Neuron.ProcessInput(FIRE);
39:       }
40:
41:       foreach(Neuron in Neurons)
42:       {
43:         Neuron.UpdateSynapses();
44:         Neuron.UpdateNeuron();
45:
46:         foreach(Synapse in Synapses)
47:         {
48:           if(Synapse.Weight < 0.0 and not Path.Contains(Synapse))
49:           {
50:             Synapse.Destroy();
51:           }
52:         }
53:       }
```

As diferenças entre v4 e v3 estão realçadas em amarelo nas linhas 15 a 22 do pseudocódigo acima. A figura A.9 ajuda a explicar essas diferenças de comportamento durante as execuções desse algoritmo.

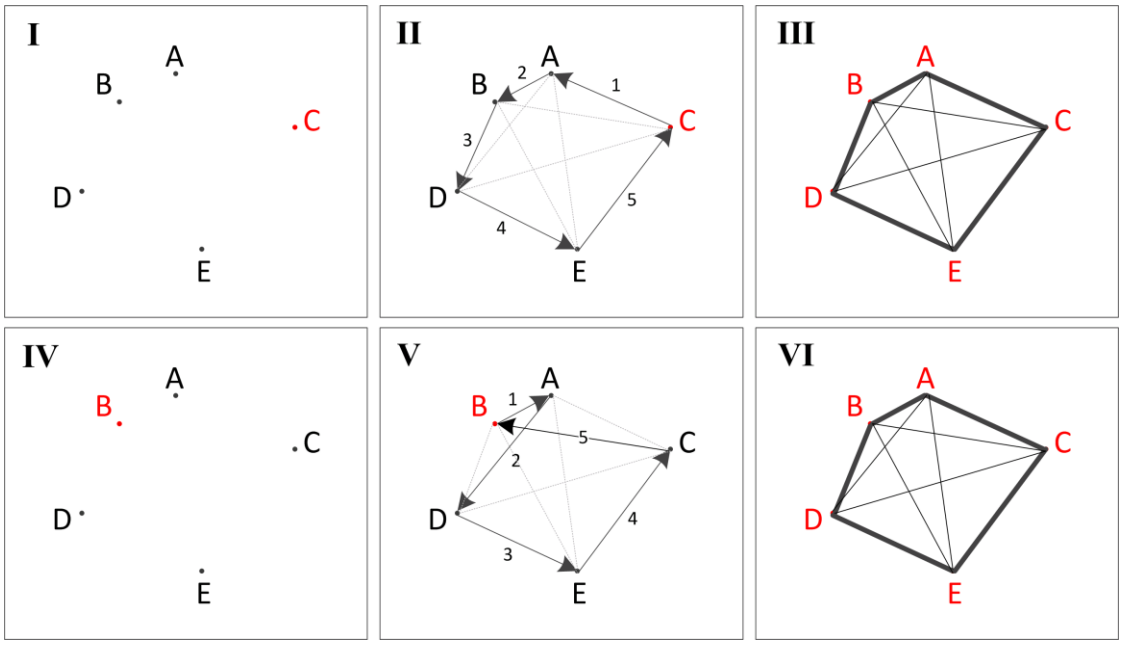

Figura A.9. A diferença em relação ao algoritmo anterior, definida pelas linhas 15 a 22 do pseudocódigo, está ilustrada nos quadros I a VI. Após a escolha da cidade inicial C, mostrado no quadro I, um caminho guloso é montado, como exibido nos quadros II e III. O quadro IV representa a escolha de B como cidade inicial em um momento seguinte, e o quadro V a construção do caminho guloso correspondente. Contudo, conforme estabelecido pelas linhas 15 a 22 do pseudocódigo, o caminho montado partindo de B é pior do que aquele montado partindo de C. Assim, o caminho partindo de B é descartado, ficando-se com o melhor caminho até então encontrado, como o quadro VI ilustra.

Pseudocódigo de um passo de tempo do algoritmo v5.

```
01:       if(NextInitialNeuron == NULL)
02:       {
03:         InitialNeuron = Neurons.GetRandomNeuron();
04:       }
05:       else
06:       {
07:         InitialNeuron = NextInitialNeuron;
08:         NextInitialNeuron = NULL;
09:       }
10:
11:       UsedNeurons.Add(InitialNeuron);
12:
13:       if(UsedNeurons.Count >= Neurons.Count)
14:       {
15:         UsedNeurons.Clear();
16:       }
17:
18:       NextNeuron = SearchNeuronGreedy(InitialNeuron, not in UsedNeurons);
19:       Path.AddLeft(NextNeuron);
20:
21:       NextInitialNeuron = NextNeuron;
22:
23:       NextNeuron = SearchNeuronGreedy(InitialNeuron, not in Path);
24:       Path.AddRight(NextNeuron);
25:
26:       if(Path.RightWeight > Path.LeftWeight)
27:       {
28:         Direction = RIGHT;
29:       }
30:       else
31:       {
32:         Direction = LEFT;
33:       }
34:
35:       while(Path.Count < Neurons.Count)
36:       {
37:         if(Direction == LEFT)
38:         {
39:           NextNeuron = SearchNeuronGreedy(InitialNeuron, not in Path);
40:           Path.AddLeft(NextNeuron);
41:         }
42:         else
43:         {
44:           NextNeuron = SearchNeuronGreedy(InitialNeuron, not in Path);
45:           Path. AddRight(NextNeuron);
46:         }
47:
48:         if(Path.RightWeight > Path.LeftWeight)
49:         {
50:           Direction = RIGHT;
51:         }
52:         else
53:         {
54:           Direction = LEFT;
55:         }
56:       }
57:
58:       if(Path.Weight > BestPath.Weight)
59:       {
60:         BestPath = Path;
61:       }
62:       else
63:       {
64:         Path = BestPath;
65:       }
66:
67:       foreach(Synapse in Synapses)
68:       {
69:         if(Path.Contains(Synapse))
70:         {
71:           Synapse.Type = EXCITATORY;
```

```
72:         }
73:         else
74:         {
75:           Synapse.Type = INHIBITORY;
76:         }
77:       }
78:
79:       foreach(Neuron in Neurons)
80:       {
81:         Neuron.ProcessInput(FIRE);
82:       }
83:
84:       foreach(Neuron in Neurons)
85:       {
86:         Neuron.UpdateSynapses();
87:         Neuron.UpdateNeuron();
88:
89:         foreach(Synapse in Synapses)
90:         {
91:           if(Synapse.Weight < 0.0 and not Path.Contains(Synapse))
92:           {
93:             Synapse.Destroy();
94:           }
95:         }
96:       }
```

As diferenças entre o algoritmo v5 e o algoritmo anterior estão destacadas em amarelo nas linhas 01 a 56 do pseudocódigo acima. A figura A.10 tenta mostrar essas diferenças.

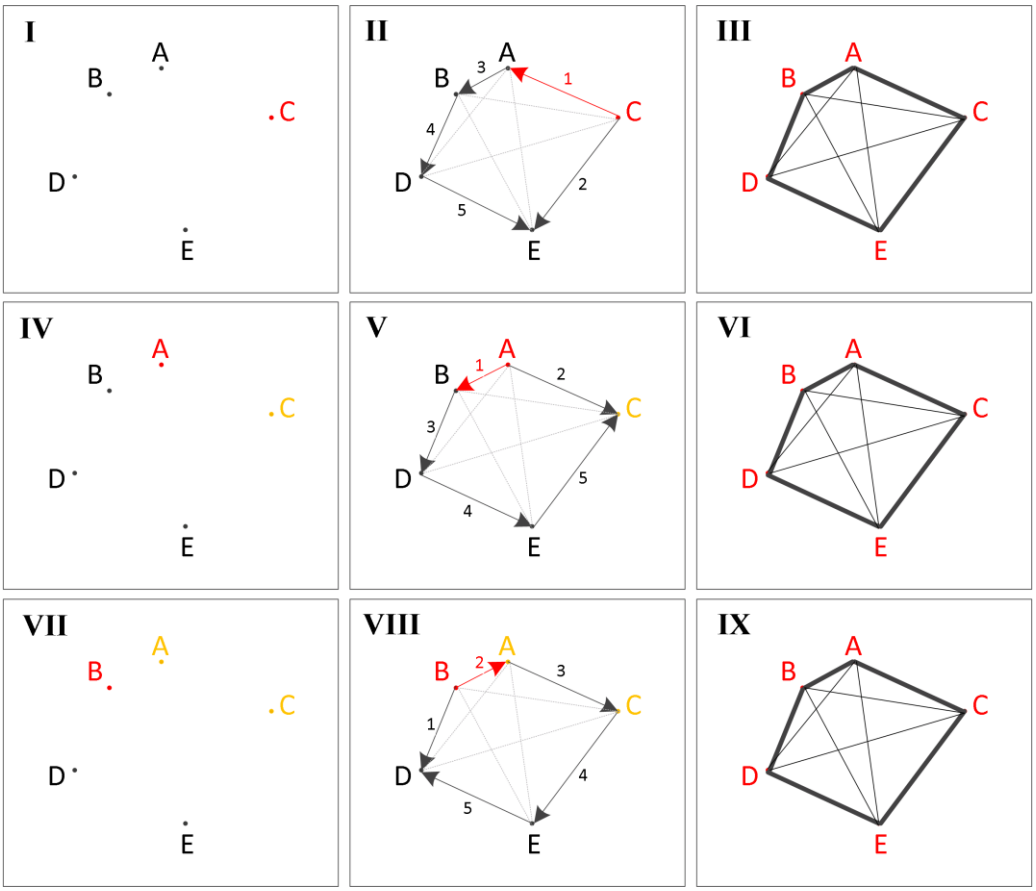

Figura A.10. A diferença em relação ao algoritmo anterior se dá, principalmente, pela mudança na busca pelo caminho guloso, conforme definido nas linhas 01 a 65 do pseudocódigo. Escolhida primeiramente de forma aleatória, a cidade inicial (assim como no algoritmo anterior) passa a fazer parte do conjunto de cidades escolhidas como iniciais (quadro I). As ligações chamadas de 1 e 2 (ou de "lado esquerdo" e "lado direito" no capítulo 3) com as duas cidades mais próximas, A e E no caso, são selecionadas para compor o caminho, conforme mostra o quadro II. Apesar de irrelevante no momento em que está vazio o conjunto de cidades escolhidas como iniciais, de acordo com a linha 18 do pseudocódigo, a ligação 1 deve obrigatoriamente levar a uma cidade ainda não escolhida como inicial (não constar no conjunto de cidades escolhidas como iniciais). A cidade conectada pela ligação 1 será a cidade inicial na interação seguinte. Partindo da cidade mais próxima (a de maior peso), um caminho guloso é montado. Assim como no algoritmo anterior, o melhor caminho até então encontrado é guardado. As ligações desse caminho são definidas como excitatórias e as demais inibitórias. Então, todas as cidades recebem estímulos e suas ligações fortalecidas ou enfraquecidas. As ligações negativas ou nulas são então eliminadas. O processo se repete seguindo pela cidade mais próxima da cidade escolhida como inicial no passo de tempo anterior, até restar um único caminho possível, sendo esse o caminho-solução. Os quadros IV a VI mostram a sequencia do algoritmo após a primeira interação. As cidades destacadas em amarelo foram escolhidas como iniciais. Os quadros VII a IX ilustram outra interação em que é possível observar o efeito da linha 18 do pseudocódigo na escolha da ligação 1. Apesar de a cidade A ser mais próxima de B do que a cidade D, a ligação 1 é definida como sendo a ligação entre A e D, pois A já foi escolhida como cidade inicial.

Pseudocódigo de um passo de tempo do algoritmo v6.

```
001:      if(NextInitialNeuron == NULL)
002:      {
003:        InitialNeuron = Neurons.GetRandomNeuron();
004:      }
005:      else
006:      {
007:        InitialNeuron = NextInitialNeuron;
008:        NextInitialNeuron = NULL;
009:      }
010:
011:      UsedNeurons.Add(InitialNeuron);
012:
013:      if(UsedNeurons.Count >= Neurons.Count)
014:      {
015:        UsedNeurons.Clear();
016:      }
017:
018:      NextNeuron = SearchNeuronGreedy(InitialNeuron, not in UsedNeurons);
019:      Path.AddLeft(NextNeuron);
020:
021:      NextInitialNeuron = NextNeuron;
022:
023:      NextNeuron = SearchNeuronGreedy(InitialNeuron, not in Path);
024:      Path.AddRight(NextNeuron);
025:
026:      if(Path.RightWeight > Path.LeftWeight)
027:      {
028:        Direction = RIGHT;
029:      }
030:      else
031:      {
032:        Direction = LEFT;
033:      }
034:
035:      while(Path.Count < Neurons.Count)
036:      {
037:        if(Direction == LEFT)
038:        {
039:          NextNeuron = SearchNeuronGreedy(InitialNeuron, not in Path);
040:          Path.AddLeft(NextNeuron);
041:        }
042:        else
043:        {
044:          NextNeuron = SearchNeuronGreedy(InitialNeuron, not in Path);
045:          Path. AddRight(NextNeuron);
046:        }
047:
048:        if(Path.RightWeight > Path.LeftWeight)
049:        {
050:          Direction = RIGHT;
051:        }
052:        else
053:        {
054:          Direction = LEFT;
055:        }
056:      }
057:
058:      if(Path.Weight > BestPath.Weight)
059:      {
060:        BestPath = Path;
061:      }
062:      else
063:      {
064:        Path = BestPath;
065:      }
066:
067:      Synapse = Path.Synapses.GetWeakest();
068:      UsedNeurons.Add(Synapse.NeuronA);
069:      UsedNeurons.Add(Synapse.NeuronB);
070:
071:
```

```
072:      foreach(Synapse in Synapses)
073:      {
074:        if(Path.Contains(Synapse))
075:        {
076:          Synapse.Type = EXCITATORY;
077:        }
078:        else
079:        {
080:          Synapse.Type = INHIBITORY;
081:        }
082:      }
083:
084:      foreach(Neuron in Neurons)
085:      {
086:        Neuron.ProcessInput(FIRE);
087:      }
088:
089:      foreach(Neuron in Neurons)
090:      {
091:        Neuron.UpdateSynapses();
092:        Neuron.UpdateNeuron();
093:
094:        foreach(Synapse in Synapses)
095:        {
096:          if(Synapse.Weight < 0.0 and not Path.Contains(Synapse))
097:          {
098:            Synapse.Destroy();
099:          }
100:        }
101:      }
```

As diferenças entre v6 e v5 estão destacadas em amarelo nas linhas 67 a 69 do pseudocódigo. A ligação mais fraca do caminho montado define o par de cidades que será ignorado na escolha da cidade inicial na interação seguinte, bem como o processo de escolha da ligação 1 (chamada de “lado esquerdo” no capítulo 3).

APÊNDICE B

Artigo publicado nos anais do congresso DINCON’09.

DINCON'09
8TH BRAZILIAN CONFERENCE ON DYNAMICS, CONTROL AND APPLICATIONS
MAY 18-22, 2009

PLASTICIDADE SINÁPTICA E O PROBLEMA DO CAIXEIRO VIAJANTE

Ricardo Fábio Sato [1], *Luiz Henrique Alves Monteiro* [1,2]

[1] Escola de Engenharia, Universidade Presbiteriana Mackenzie
[2] Escola Politécnica, Universidade de São Paulo

Emails: rfsato@gmail.com; luizhenrique.monteiro@mackenzie.br, luizm@usp.br

Resumo: Propõem-se dois algoritmos, inspirados no fato de que sinapses que conectam neurônios biológicos podem se alterar com o passar do tempo em função da atividade desses neurônios, para solucionar o clássico problema de otimização chamado de o problema do caixeiro viajante. O desempenho dos algoritmos aqui propostos é avaliado comparando-se o comprimento do caminho gerado por eles com os comprimentos de todos os caminhos possíveis. Os resultados obtidos indicam que um desses algoritmos parece ser uma abordagem bastante promissora: para o caso em que o caixeiro deve passar por 10 cidades, a solução ótima foi encontrada em 79% das simulações realizadas.

Palavras-chave: *lei de Hebb, plasticidade sináptica, problema do caixeiro viajante.*

1. INTRODUÇÃO

O problema do caixeiro viajante é um exemplo clássico de problema de otimização combinatória, pertencente à classe NP-completo, que vem sendo estudado há pelo menos 150 anos [1]. Sua formulação é a seguinte. Considere N cidades e o custo de ir de uma cidade para outra. O custo pode ser, por exemplo, a distância, o gasto com combustível ou o tempo de viagem (se o custo de ir da cidade a para a cidade b é diferente do custo de se ir de b para a, tem-se o caso assimétrico [2]; aqui, trata-se do caso simétrico, no qual esses custos são iguais). O objetivo é determinar o percurso fechado que visita cada cidade uma única vez com custo mínimo e que retorna à cidade de partida. Determinar a solução ótima através de busca exaustiva é uma tarefa computacionalmente inviável se N é grande (nesse contexto, "grande" significa da ordem de 20 ou maior), dado que o número de caminhos possíveis vale $(N-1)!$. Por exemplo, para $N=20$, há mais de $1{,}2\times10^{7}$ percursos possíveis e encontrá-los consumiria um tempo de processamento da ordem de meses num computador pessoal típico dos dias de hoje.

Existem inúmeros algoritmos que procuram resolver, ao menos aproximadamente [3], esse problema, como redes neurais artificiais de Hopfield [4], algoritmos baseados em colônias de formigas [5] e algoritmos genéticos [6]. Normalmente, a solução encontrada é aproximada (sub-ótima).

Aqui, propõem-se dois algoritmos para solucionar o problema do caixeiro viajante, baseados em regras de modificação sináptica encontradas em redes neurais biológicas. Assume-se que cada cidade corresponde a um neurônio da rede e que a intensidade da sinapse que conecta dois neurônios é igual ao inverso do custo de ir de uma cidade para outra. Desse modo, as sinapses de maiores intensidades correspondem aos caminhos de menores custos.

Este trabalho está assim organizado. Na seção 2, trata-se de plasticidade sináptica, e relaciona-se a malha viária que conecta um conjunto de cidades, a ser percorrida de forma ótima pelo caixeiro, com uma rede de conexões sinápticas que liga um conjunto de neurônios. Os algoritmos propostos são descritos na seção 3 e relatam-se seus desempenhos na seção 4. Na seção 5, apresentam-se as principais conclusões.

2. PLASTICIDADE SINÁPTICA

O funcionamento (normal ou patológico) de um sistema nervoso biológico depende fortemente da topologia da rede neural correspondente, já que ela afeta o desempenho de funções típicas desse sistema, como aprendizagem e memória [7]. Além disso, sabe-se que as intensidades das sinapses de uma rede neural biológica se modificam com o passar do tempo, em função da presença ou ausência de estímulos [8]. A essa capacidade de modificação das sinapses dá-se o nome de plasticidade sináptica [9, 10].

Segundo Hebb, se um neurônio provoca o disparo de outro, então há o fortalecimento da sinapse entre ambos [11]. Essa proposição, conhecida como lei de Hebb, apóia-se no conceito de sincronismo (ou talvez seja mais preciso dizer, na correlação) entre disparos neurais. O estudo de Hebb tratava de sinapses excitatórias, mas sinapses inibitórias parecem se comportar de modo semelhante; ou seja, elas se intensificam se há o disparo síncrono entre os neurônios pré e pós-sináptico e se enfraquecem caso contrário [12]. Sinapses não estimuladas tendem a se enfraquecer e podem chegar a desaparecer. Esses fatos são fundamentais para a compreensão dos nossos algoritmos.

Neste trabalho, faz-se um paralelo entre cidades conectadas por uma malha viária e os custos associados às viagens entre essas cidades, e neurônios conectados por uma rede de ligações sinápticas e as intensidades (os pesos) associadas a essas sinapses.

A rede neural artificial, que corresponde à malha viária a ser percorrida pelo caixeiro, é assim modelada. Os *N* neurônios (as *N* cidades) que formam a rede são distribuídos aleatoriamente num reticulado quadrado de dimensão *L*×*L*, de modo que cada célula desse reticulado pode ser ocupada por apenas um único neurônio. Inicialmente todos os neurônios são conectados entre si. As conexões entre eles (caminhos) correspondem às sinapses existentes entre neurônios biológicos.

O conjunto de todas as ligações entre os neurônios determina a topologia de acoplamento da rede neural. As conexões são consideradas não-direcionadas; ou seja, a sinapse que parte do neurônio *a* e chega ao neurônio *b* corresponde à mesma sinapse que liga o neurônio *b* ao neurônio *a*. E não existem sinapses que conectam um neurônio a ele próprio; isto é, nessa rede neural não há auto-conexões.

Como ponto de partida, cria-se, então, um espaço delimitado por uma matriz quadrada *L*×*L*. Nesse espaço, *N* neurônios são distribuídos de forma aleatória e cada neurônio é ligado a todos os demais, como ilustrado na figura 1 para um caso em que *L=10* e *N=5*. Ou seja, numa matriz *10×10*, distribuem-se *5* cidades aleatoriamente e, inicialmente, há caminhos conectando todos pares de cidades.

A intensidade de cada sinapse é igual ao inverso do valor da distância euclidiana entre o par de neurônios que ela conecta. Portanto, quanto mais próximo um neurônio está de outro ao qual está conectado, mais forte é a sinapse (maior é o peso sináptico) que os liga.

Na próxima seção, descrevem-se os dois algoritmos propostos para solucionar o problema de caixeiro viajante.

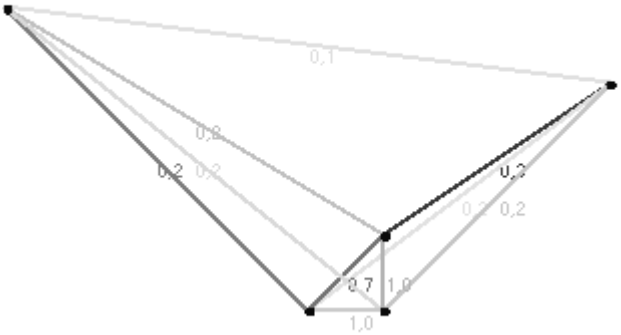

Figura 1. Exemplo de rede neural criada em uma matriz *10×10* na qual as posições de *5* neurônios foram sorteadas aleatoriamente. Os círculos escuros correspondem a esses neurônios e os números ao lado das conexões indicam os valores dos pesos sinápticos (esses números estão representados com apenas um algarismo significativo, por questão de clareza da figura). Note que caminhos mais compridos correspondem a pesos menores.

3. OS ALGORITMOS PROPOSTOS

Como mencionado, cada neurônio da rede neural corresponde a uma cidade a ser visitada, cada sinapse representa o caminho entre duas cidades, e o peso sináptico vale o inverso da distância entre as cidades. Foram criados dois algoritmos para resolver o problema, chamados simplesmente de I e II.

O algoritmo I é determinista, e, de acordo com esse algoritmo, a busca pela solução do problema do caixeiro viajante ocorre da seguinte maneira:

- Todas as ligações existentes são enfraquecidas a cada passo de tempo, sendo o peso de cada sinapse diminuído da constante η (pois, como os neurônios não são estimulados, as intensidades das sinapses que os conectam apenas diminuem com o passar do tempo). As ligações são enfraquecidas até atingir o valor de saturação $-\varphi$ (ou seja, $-\varphi$ é o menor valor que um peso sináptico pode ter). As constantes η e φ são positivas. Note que o valor mínimo dos pesos sinápticos é $-\varphi$ (e não zero).

- No passo de tempo *t*, constrói-se um possível caminho-solução para o problema fazendo uma busca local da seguinte forma: partindo do neurônio mais próximo do canto superior esquerdo da matriz *L*×*L* (em caso de empate, parte-se do neurônio mais próximo da borda superior), caminha-se até o neurônio que está ligado a esse neurônio de partida pelo maior peso sináptico (que corresponde ao caminho de menor distância). Então, vai-se desse segundo neurônio até um terceiro neurônio, que é aquele que está ligado ao segundo pelo maior peso sináptico. Esse procedimento é repetido até se formar um caminho que passa por todos os neurônios, uma única vez, e que retorna ao ponto de partida.

- Cria-se uma lista em que os neurônios são dispostos em ordem decrescente pela quantidade de ligações que cada um deles possui nesse passo de tempo *t*. Dessa maneira, o neurônio com o maior número de ligações figura na primeira posição e aquele com o menor número ocupa a última posição.

- Essa lista ordenada é varrida, do topo para a base, fixando um neurônio e com ele formando um par, de modo a se percorrer todos os pares da lista. A figura 2 ilustra como é feita essa varredura. Nesse passo de tempo *t*, as ligações existentes entre cada par de neurônios com pesos de valor menor ou igual a zero são eliminadas, desde que tais ligações não sejam utilizadas no caminho-solução montado anteriormente. Se uma ligação faz parte desse caminho-solução, ela é mantida, independente do valor do seu peso sináptico (que pode ser negativo, nulo ou positivo).

Dessa forma, sinapses vão sendo eliminadas com o passar do tempo, diminuindo os possíveis caminhos que compõem a solução do problema. Quando o caminho traçado se torna o único possível, já que todos os demais foram eliminados, ele é considerado a solução do problema.

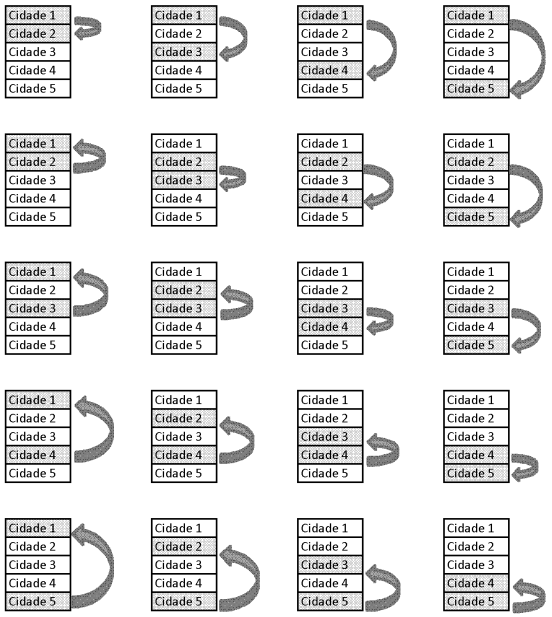

Figura 2. A varredura da lista ordenada é feita de cima para baixo e da esquerda para a direita, considerando pares de cidades (neurônios). Suponha que, nessa lista, a cidade com mais ligações é a 1, seguida pela 2, pela 3, pela 4 e, finalmente, pela cidade 5. Varre-se essa lista assim: fixa-se a cidade 1 e analisam-se as sinapses que ela faz com as demais cidades; depois fixa-se a cidade 2 e repete-se o procedimento. Finaliza-se a varredura analisando as sinapses da cidade 5 com as outras quatro cidades. Só são eliminadas as ligações (sinapses) com peso menor ou igual a zero e que não fazem parte do caminho-solução.

Esse algoritmo determinista, apesar de produzir bons resultados (como mostrado na próxima seção), tem um defeito muito grave: o caminho-solução encontrado no primeiro passo de tempo é aquele que o algoritmo fornece como solução final. Ou seja, com esse algoritmo, apenas eliminam-se as sinapses que não fazem parte do caminho-solução encontrado na primeira iteração através de uma busca local.

A fim de se obterem diferentes caminhos-solução ao longo da simulação, e com isso se chegar a resultados ainda melhores, modificou-se o algoritmo I. O algoritmo II, que possui uma componente aleatória, funciona da seguinte maneira:

- Como no algoritmo I, a cada passo de tempo, as sinapses inicialmente existentes começam a ser enfraquecidas, diminuindo-se o valor dos respectivos pesos da constante η.
- Um neurônio é selecionado aleatoriamente para servir de ponto de partida para a construção do caminho-solução (observe que no algoritmo I esse ponto de partida é fixo). Esse caminho é determinado como no algoritmo I; entretanto, ele só é aceito se a soma dos seus pesos for maior quando comparada à soma dos pesos do caminho-solução encontrado no passo de tempo anterior (isto é, ele só é adotado caso seja mais curto). Obviamente, no primeiro passo de tempo, essa comparação não é feita.
- De acordo com a lei de Hebb, se dois neurônios disparam simultaneamente, a ligação entre eles se fortalece; caso contrário, se enfraquece. No algoritmo II, impõe-se que cada vez que uma sinapse é eliminada, os neurônios conectados por ela são estimulados e, conseqüentemente, disparam (os estímulos aplicados são de intensidade maior do que o limiar de disparo dos neurônios). Em decorrência de tais estímulos, algumas das sinapses remanescentes são fortalecidas e isso se dá somando-se a constante positiva μ aos pesos das sinapses que conectam neurônios que dispararam simultaneamente (isto é, nesse mesmo passo de tempo). Se estimulado, um neurônio permanece no estado ativo por α passos de tempo e, em seguida, passa para o estado refratário (no qual não reage a novos estímulos) permanecendo assim por β passos de tempo, antes de voltar ao estado de repouso (no qual dispara, se for estimulado). Assume-se que o valor máximo que uma sinapse pode ter é φ e que o valor mínimo, como no algoritmo I, vale $-\varphi$.
- Como no algoritmo I, monta-se uma lista em que os neurônios são classificados em ordem decrescente pela quantidade de ligações que possuem naquele passo de tempo, sendo essa lista varrida conforme já explicado e ilustrado pela figura 2. Assim, sinapses vão sendo eliminadas até sobrar um único caminho, que é a solução do problema gerada pelo algoritmo II.

4. EXPERIMENTOS NUMÉRICOS

As simulações foram realizadas em um computador equipado com processador Intel Xeon Quad-core E5410 com *clock* de 2,33 GHz, 2 Gb de memória RAM, sistema operacional Microsoft Windows Server 2003 R2 Standard x64. Os algoritmos foram escritos na linguagem de programação C#, através da ferramenta Microsoft Visual Studio 2008, com a plataforma de desenvolvimento .NET Framework v2.

Em cada simulação, o que corresponde a uma nova disposição espacial das cidades a serem visitadas, todos os caminhos possíveis foram determinados, a fim de permitir a comparação entre o comprimento deles e o comprimento do

caminho-solução encontrado pelos algoritmos. Assim, os pesos das ligações entre cada par de cidades que formam esses caminhos foram somados, para se calcular o peso total *W* de tais caminhos.

Vale novamente ressaltar que o número de caminhos distintos possíveis vale *S=(N-1)!* e que quanto maior *W* mais curto é o caminho.

Os caminhos possíveis foram ordenados de forma decrescente pelo valor de *W*, de tal maneira que o melhor (mais curto) caminho aparece na posição 1 e o pior (mais comprido) caminho ocupa a posição *S*. Então, o caminho-solução gerado pelos algoritmos foi localizado nessa lista. Desse modo, o desempenho dos algoritmos foi avaliado.

Como primeiro teste, foram realizadas *200* simulações com o algoritmo I em que se distribuíram *5* neurônios aleatoriamente em um reticulado com *10×10* posições possíveis. Um exemplo é aquele mostrado na figura 1. Para uma simulação em particular com *N=5*, existem *S=24* caminhos possíveis. Em cada simulação, executou-se o algoritmo por *100* passos de tempo, tomando-se a constante de decremento sináptico $\eta=0,01$ e a constante de saturação $\varphi=2$. A figura 3 exibe o caminho-solução (que é o ótimo) encontrado pelo algoritmo I correspondente à figura 1.

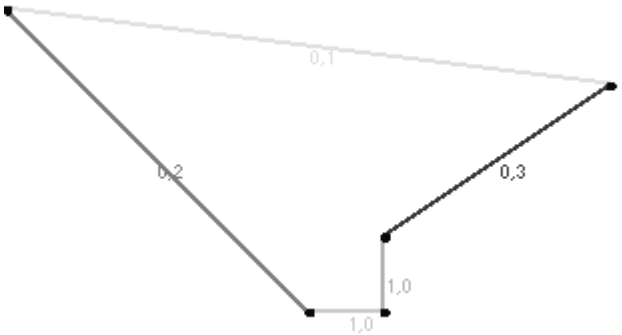

Figura 3. Caminho ótimo encontrado pelo algoritmo I para a rede mostrada na figura 1.

Tabela 1. Resultados utilizando o algoritmo I com *N=5*. A primeira coluna dessa tabela indica a posição que um caminho fechado (que resolve o problema do caixeiro viajante) ocupa na lista de posições ordenadas de modo decrescente pelo valor de *W* (ou seja, do caminho mais curto para o mais comprido). A segunda coluna apresenta a porcentagem de soluções encontradas pelo algoritmo I que ocupam tal posição.

Posição	%
1	*40%*
2 a 5	*30%*
6 a 15	*30%*
16 e 24	*0%*

Nas *200* simulações realizadas com o algoritmo I, os caminhos-solução encontrados ficaram posicionados conforme descrito pela tabela 1. Portanto, em *40%* das simulações, o algoritmo I encontrou a solução-ótima.

A fim de testar o desempenho do algoritmo I em situações mais complicadas, foram realizadas *200* simulações com *N=10* e *L=10*. Assim, preservou-se o tamanho do reticulado em relação ao primeiro conjunto de testes, mas aumentou-se o número de neurônios. Nesse caso, há *S=362880* caminhos fechados possíveis para cada simulação em particular (para cada disposição espacial de *10* cidades). Como no primeiro conjunto de testes, em cada simulação, executou-se o algoritmo I por *100* passos com $\eta=0,01$ e $\varphi=2$. A figura 4 mostra a topologia inicial de um caso a ser resolvido pelo algoritmo I, e a figura 5 o caminho-solução (que é o ótimo) encontrado por ele.

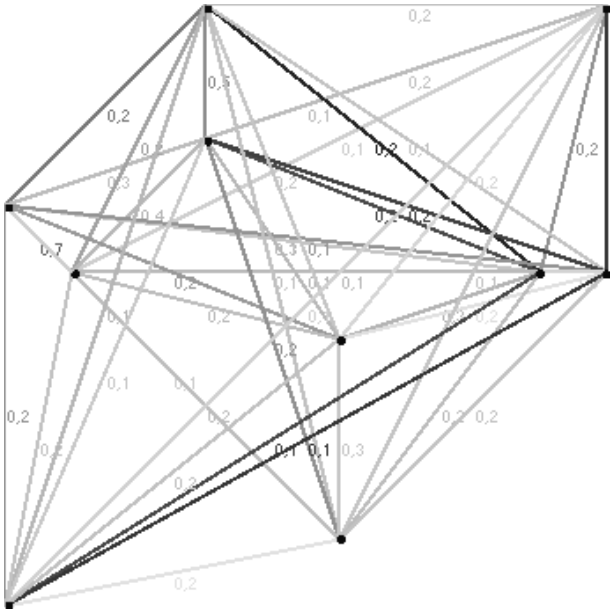

Figura 4. Rede criada em uma matriz *10×10* com *10* neurônios.

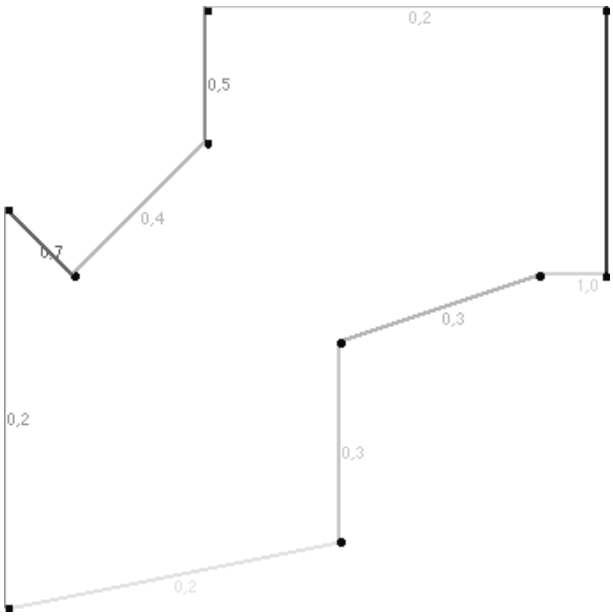

Figura 5. Caminho ótimo encontrado pelo algoritmo I para a rede mostrada na figura 4.

Nessa série de *200* simulações realizadas com *10* neurônios, os caminhos-solução encontrados pelo algoritmo I ficaram posicionados conforme descrito pela tabela 2. Portanto, em *21,5%* dos casos, esse algoritmo determinista encontrou a solução-ótima e em *87,5%* dos casos, o algoritmo I encontrou soluções localizadas nas *100* primeiras posições da lista de caminhos mais curtos, de um total de *362880* posições.

Tabela 2. Resultados utilizando o algoritmo I com *N=10*. As colunas dessa tabela têm o mesmo significado dado na legenda da tabela 1.

Posição	%
1	21,5%
2 a 5	17%
6 a 15	21%
16 a 100	28%
101 a 1000	11,5%
1001 a 10000	1%
10001 a 362880	0%

O desempenho do algoritmo II foi avaliado em situações similares. Primeiro, foram realizadas *200* simulações com *5* neurônios distribuídos aleatoriamente em um reticulado com *10×10* posições possíveis, executando o algoritmo por *200* passos de tempo, e tomando-se a constante de decremento sináptico $\eta=0,01$; a constante de incremento sináptico $\mu=0,02$; e a constante de saturação $\varphi=1$. As constantes relacionadas à duração dos estados ativo e refratário foram escolhidas como $\alpha=1$ e $\beta=2$, respectivamente. A tabela 3 mostra os resultados das simulações.

Tabela 3. Resultados utilizando o algoritmo II com *N=5*. As colunas dessa tabela têm o mesmo significado dado na legenda da tabela 1.

Posição	%
1	99,5%
2 a 5	0,5%
6 a 15	0,0%
16 a 24	0,0%

Em *99,5%* dos casos, o algoritmo II encontrou a solução-ótima, sendo que em *100%* dos casos as soluções localizadas estavam dentre as 5 primeiras posições da lista de caminhos mais curtos, de um total de *24* posições.

Ainda, com o algoritmo II, foram realizadas *200* simulações com *N=10* e *L=10*. Em cada simulação o algoritmo foi executado por *200* passos com $\eta=0,01$; $\mu=0,02$; $\varphi=1$; $\alpha=1$ e $\beta=2$. Os resultados estão mostrados na tabela 4.

Tabela 4. Resultados utilizando o algoritmo II com *N=10*. As colunas dessa tabela têm o mesmo significado dado na legenda da tabela 1.

Posição	%
1	79,0%
2 a 5	15,5%
6 a 15	4,0%
16 a 100	1,5%
101 a 1000	0,0%
1001 a 10000	0,0%
10001 e abaixo	0,0%

Para *79%* dos casos, o algoritmo II encontrou a solução-ótima, sendo que em *100%* dos casos, o algoritmo encontrou soluções localizadas nas *100* primeiras posições da lista de caminhos mais curtos, de um total de *362880* posições.

A figura 6 ilustra a topologia de uma rede com *N=10* e com, inicialmente, todos os neurônios ligados entre si; a figura 7 exibe o caminho-solução ótimo gerado pelo algoritmo II.

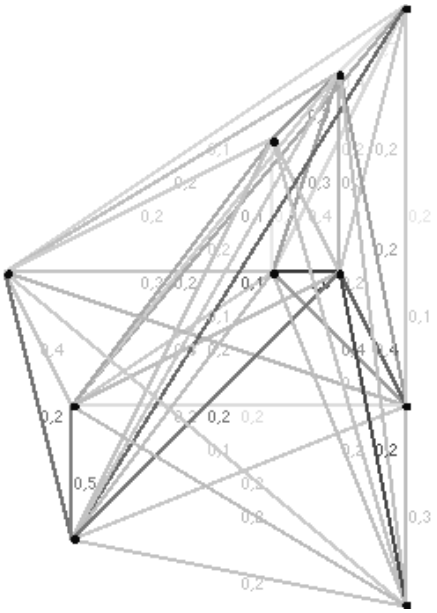

Figura 6. Rede criada em uma matriz *10×10* com *10* neurônios.

Fatores que afetam fortemente o desempenho desse algoritmo são os valores das constantes η, μ, φ, α e β. Não sabemos, no momento, como escolher os "melhores" valores para essas constantes.

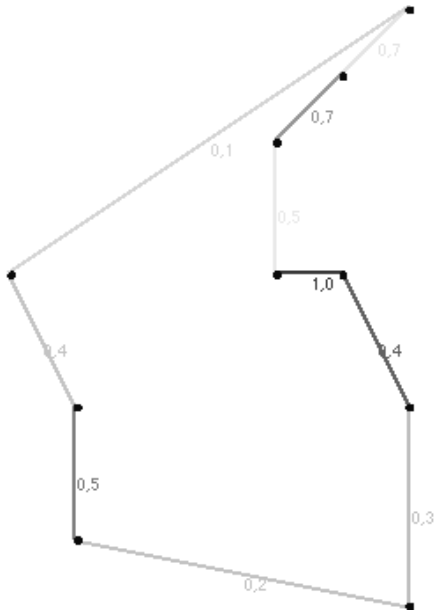

Figura 7. Caminho ótimo encontrado pelo algoritmo II para a rede mostrada na figura 6.

Para *N=10*, o algoritmo II gasta em torno de 1,2 segundos para ser executado no computador citado (o I é ainda mais rápido); enquanto que a geração de todos os caminhos possíveis, necessários para se encontrar a solução ótima por força-bruta, consome 3,5 horas.

5. CONCLUSÕES

Esses algoritmos simples, baseados nas modificações sinápticas que ocorrem em redes neurais biológicas, apresentaram um desempenho que pode ser considerado satisfatório, principalmente para o caso mais difícil estudado neste trabalho, que é aquele com *10* cidades (neurônios). O desempenho do algoritmo II foi bastante superior ao do algoritmo I, e os resultados mostrados nas tabelas 3 e 4 sugerem que essa é uma abordagem promissora. Aqui, estudou-se apenas o problema do caixeiro viajante simétrico; mas o caso assimétrico poderia ser tratado de modo similar.

Como próximos passos, além de investigar maneiras de escolher "bons" valores para as constantes η, μ, φ, α e β, pretende-se introduzir novas componentes aleatórias no algoritmo II, a fim de tentar evitar a convergência para soluções sub-ótimas; e realizar testes em malhas formadas por um maior número de cidades. A questão da complexidade computacional dos algoritmos desenvolvidos está também ainda sem resposta.

AGRADECIMENTO

L. H. A. Monteiro agradece o suporte financeiro do CNPq.

REFERÊNCIAS

[1] S. Lin e B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem", Operations Research, V. 21, pp. 498-516, 1973.

[2] T. Oncana, I. K. Altınelb e G. Laportec, "A comparative analysis of several asymmetric traveling salesman problem formulations", Computers and Operations Research, V. 36, pp. 637-654, 2009.

[3] T. P. Bagchi, J. N. D. Gupta e C. Sriskandarajah, "A review of TSP based approaches for flowshop scheduling", European Journal of Operational Research, V. 169, pp. 816-854, 2006.

[4] C. Buosoño-Calzón e A. R. Figueiras-Vidal, "A bank of Hopfield neural networks for the shortest path problem", Signal Processing, V. 61, pp. 157-170, 1997.

[5] M. Dorigo, V. Maniezzo e A. Corloni, "The ant system: optimization by a colony of cooperating agents", IEEE Transactions on Systems, Man, and Cybernetics, V. 26, pp. 29-41, 1996.

[6] J. Y. Potvin, "Genetic algorithms for the traveling salesman problem", Annals of Operations Research, V. 63, pp. 339-370, 1996.

[7] M. F. Bear, B. W. Connors e M. A. Paradiso, "Neurociências", 2ª ed., Artmed Editora, 2002.

[8] D. V. Buonomano e M. M. Merzenich, "Cortical plasticity: from synapses to maps", Annual Review of Neuroscience, V. 21, pp. 149-186, 1998.

[9] S. Song e L. F. Abbott, "Cortical development and remapping through spike timing-dependent plasticity", Neuron, V. 32, pp. 339-350, 2001.

[10] R. A. Menezes e L. H. A. Monteiro, "Investigando o desempenho da rede de Hopfield com mal de Alzheimer", Proceedings of the 7th Brazilian Conference on Dynamics, Control and Applications, V. 1, 589-593, 2008.

[11] S. J. Cooper, "Donald O. Hebb's synapse and learning rule: a history and commentary", Neuroscience and Biobehavioral Reviews, V. 28, pp. 851-874, 2005.

[12] B. Siri, M. Quoy, B. Delord, B. Cessac e H. Berry, "Effects of hebbian learning on the dynamics and structure of random networks with inhibitory and excitatory neurons", Journal of Physiology (Paris) V. 101, pp. 136-148. 2007.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo