UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

VINÍCIUS MOURA GIGLIO

Aplicação de algoritmos bioinspirados na definição da trajetória ótima de pulsos ultrassônicos em elementos de concreto

São Carlos
2021

VINÍCIUS MOURA GIGLIO

Aplicação de algoritmos bioinspirados na definição da trajetória ótima de pulsos ultrassônicos em elementos de concreto

VERSÃO CORRIGIDA

A versão original encontra-se na Escola de Engenharia de São Carlos

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Civil (Engenharia de Estruturas) da Escola de Engenharia de São Carlos da Universidade de São Paulo para obtenção do título de Mestre em Ciências.

Área de Concentração: Estruturas.

Orientador: Prof. Dr. Vladimir Guilherme Haach.

São Carlos

2021

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Giglio, Vinícius Moura
G459a Aplicação de algoritmos bioinspirados na definição da trajetória ótima de pulsos ultrassônicos em elementos de concreto / Vinícius Moura Giglio; orientador Vladimir Guilherme Haach. São Carlos, 2021.

Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia Civil (Engenharia de Estruturas) e Área de Concentração em Estruturas -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2021.

1. Tomografia. 2. Ultrassom. 3. Algoritmos Genéticos. 4. Colônia de Formigas. 5. Ensaios não destrutivos. 6. Concreto. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE JULGAMENTO

Candidato: Engenheiro **VINICIUS MOURA GIGLIO.**

Título da dissertação: "Aplicação de algoritmos bioinspirados na definição da trajetória ótima de pulsos ultrassônicos em elementos de concreto".

Data da defesa: 03/08/2021.

Comissão Julgadora	**Resultado**
Prof. Associado **Vladimir Guilherme Haach** **(Orientador)** (Escola de Engenharia de São Carlos – EESC/USP)	*Aprovado*
Profa. Dra. **Maria Cecilia Amorim Teixeira da Silva** (Universidade Estadual de Campinas/UNICAMP)	*Aprovado*
Prof. Dr. **Lourenço Panosso Perlin** (Universidade Federal de Santa Catarina/UFSC)	*Aprovado*

Coordenador do Programa de Pós-Graduação em Engenharia Civil (Engenharia de Estruturas):
Prof. Associado **Vladimir Guilherme Haach**

Presidente da Comissão de Pós-Graduação:
Prof. Titular **Murilo Araujo Romero**

AGRADECIMENTOS

Agradeço ao Prof. Dr. Vladimir Guilherme Haach pela atenção e disponibilidade constante ao longo do processo de orientação, além do apoio, boas ideias e opiniões sinceras sempre que precisei.

Aos docentes da UNESP Ilha Solteira, onde me graduei, pelos ensinamentos e por despertarem meu interesse pela área de estruturas.

Aos docentes do Departamento de Engenharia de Estruturas (SET) da EESC-USP que contribuíram para a minha formação como mestre e para a realização deste trabalho de alguma forma, especialmente os docentes Prof. Dr. Rogério Carrazedo e Prof. Dr. Edson Denner Leonel, presentes em minha banca de qualificação.

À Escola de Engenharia de São Carlos (EESC-USP) pela oportunidade de realização do curso de mestrado.

Aos colegas de turma do mestrado Andressa, Caio, Daniel, Danilo, Darcy, Letícia e Luiz Fernando agradeço pela convivência, ajuda nas disciplinas e por todos os momentos que vivemos juntos ao longo destes dois anos.

Aos meus pais, Márcio e Dalva, por todo o esforço que fizeram e fazem por mim e a meu irmão, Leonardo, pela parceria de sempre. Sem vocês eu não estaria aqui.

Também gostaria de agradecer minha namorada, Fernanda, pelo amor e apoio em todos os momentos em que precisei e pela compreensão nos momentos em que estive ausente para me dedicar ao trabalho.

Por fim, gostaria de agradecer aos meus amigos de Luziânia/GO, do Ensina Brasil, pessoas incríveis que conheci e que se tornaram tão especiais em pouco tempo, pelo apoio no processo de escrita do fim do trabalho e pelos melhores desejos para o meu futuro.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Código de financiamento 001.

RESUMO

GIGLIO, V. M. **Aplicação de algoritmos bioinspirados na definição da trajetória ótima de pulsos ultrassônicos em elementos de concreto.** 2021. 157p. Dissertação (Mestrado em Ciências – Engenharia Civil (Estruturas)) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2021.

O ensaio de ultrassom tem se mostrado muito eficiente na detecção de heterogeneidades em estruturas e consiste na alocação de transdutores piezoelétricos em um elemento estrutural, que geram pulsos. Com as medições de tempo de viagem dos pulsos e as posições dos transdutores, calcula-se a velocidade dos pulsos. Aliando o ensaio à técnica de tomografia computadorizada, é possível gerar imagens do mapa de velocidades do pulso em seções da estrutura. À partir da comparação das velocidades, é possível identificar heterogeneidades. No entanto, para calcular as velocidades, é necessário que se assuma uma hipótese a respeito da trajetória seguida pelo pulso, geralmente suposta retilínea. O presente trabalho objetiva abandonar esta hipótese, determinando as trajetórias dos pulsos com a utilização de algoritmos determinísticos (Dijkstra e A*) e bioinspirados (algoritmos genéticos e otimização por colônia de formigas). Os algoritmos foram implementados no *software* de geração de imagens em estruturas TUSom e testados em duas seções de concreto com defeitos simulados. Os resultados permitiram classificar os algoritmos de acordo com a qualidade das soluções e tempo de processamento. As novas trajetórias indicaram um comportamento mais fiel à realidade, em que os pulsos desviaram das heterogeneidades. Assim, pode-se afirmar que o mapeamento foi bem-sucedido. Com relação aos melhores algoritmos, em ordem decrescente de qualidade, obtiveram-se: Dijkstra v2 (nova implementação), A*, ACS, GA e Dijkstra v1 (implementação inicial). Portanto, o algoritmo Dijkstra v2 será implementado como a alternativa padrão para resolução do problema das trajetórias no *software* TUSom. Comparando o pior e o melhor algoritmo no exemplo 2, mais complexo, os tempos de processamento diminuíram de cerca 18 horas para menos de 20 minutos. Apesar dos algoritmos bioinspirados não apresentarem os melhores desempenhos, como era esperado, se mostraram alternativas com potencial, especialmente o ACS.

Palavras-chave: tomografia; ultrassom; algoritmos genéticos; colônias de formigas; ensaios não destrutivos; concreto.

ABSTRACT

GIGLIO, V. M. **Application of bioinspired algorithms in the determination of ultrassonic wave optimal paths in concrete elements.** 2021. 157p. Dissertation (M. Sc. in Civil Engineering (Structural Engineering)) – Sao Carlos School of Engineering, University of Sao Paulo, Sao Carlos, 2021.

The ultrasonic pulse velocity test (UPV) has shown consistent results in integrity monitoring of structures. The test equipment consists in a set of two piezoeletric transducers that generate stress waves and measure the pulse travel time inside the structure. With travel time and the distance between transducers it is possible to compute the ultrassonic pulse velocity. Combining UPV and computerized tomography it is feasible to create velocity maps of the structure cross-sections and find defects. However, to determine the ultrasonic pulse velocity it is necessary to assume a hypothesis about the wave paths, usually considered as straight lines. This work intends to abandon this idea and to determine the ultrasonic wave paths using deterministic (Dijkstra and A*) and bioinspired algorithms (genetic algorithms and ant colony optimization). To reach this goal, the algorithms were implemented in the software of tomographic imaging in structures TUSom and tested in two concrete cross-sections with simulated defects. The results allowed to classify the best algorithm according with two criterias: solution quality and processing time. The pulses dodged the simulated defects, which indicates that the new wave paths represented a more realistic behavior. Thus, it is possible to say that the wave path mapping was successful. In the end, Dijkstra v2 (new implementation) was the best algorithm, followed by A*, ACS, GA and Dijkstra v1 (old implementation). Therefore, the Dijkstra v2 algorithm will be implemented as the standard procedure to determine the ultrasonic wave paths in TUSom software. Comparing the worst and the best algorithms, the processing time dropped from about from 5 hours and 30 minutes to less than 10 minutes in the first example and from approximately 18 hours to less than 20 minutes in the second example. Contrary to expectations, the bioinspired algorithms did not present the best performances. Despite that, they arise as alternatives with great potential, especially the ACS algorithm.

Keywords: tomography; ultrasound; genetic algorithms; ant colony optimization; nondestructive tests; concrete.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ACO	*Ant Colony Optimization*
ACS	*Ant Colony System*
ART	*Algebraic Reconstruction Technique*
AS	*Ant System*
CL	*Candidate Lists*
DNA	Ácido Desoxirribonucleico
EAS	*Elitist Ant System*
EESC	Escola de Engenharia de São Carlos
END	Ensaio Não Destrutivo
FPC	*Free Pascal Compiler*
GA	*Genetic Algorithm*
MMAS	*MAX-MIN Ant System*
NPP	*Nondeterministic Polynomial Problems*
OLS	*Ordinary Least Squares*
PC	Probabilidade de Cruzamento
PM	Probabilidade de Mutação
PP	*Polynomial Problems*
PSO	*Particle Swarm Optimization*
RNA	Redes Neurais Artificiais
SET	Departamento de Engenharia de Estruturas da EESC
SIRT	*Simultaneous Iteractive Reconstruction Technique*
TC	Tomografia Computadorizada
TSP	*Traveling Salesman Problem*
UPV	*Ultrasonic Pulse Velocity Test*
USP	Universidade de São Paulo

SUMÁRIO

1 INTRODUÇÃO

O concreto é um dos materiais mais utilizados pela humanidade. A sua grande resistência à compressão, aliada a baixo custo e cada vez mais evolução no campo de estudo dos materiais, contribuíram muito para a sua popularização. Hoje, é possível a construção de qualquer tipo de obra, desde pequenas casas à barragens, pontes e aeroportos.

Com o passar do anos, no entanto, são cada vez mais comuns as obras de restauração, visto que as estruturas são projetadas com determinada vida útil. Também é comum a necessidade de reforço estrutural tendo como causa a má execução ou o surgimento de parâmetros não previstos em projeto, como a alteração da utilização da construção. Além disso, obras de maior porte exigem monitoramento constante de sua integridade estrutural.

Uma ferramenta muito eficiente e que tem conquistado visibilidade na área são os ensaios não destrutivos, úteis na detecção de defeitos, alteração de propriedades ou fissuração do concreto e que permitem ensaios rápidos e menos invasivos. Uma grande vantagem desse tipo de ensaio é a possibilidade de avaliar a mesma estrutura inúmeras vezes, minimizando a variabilidade estatística (presente em ensaios destrutivos convencionais) e permitindo a análise de parâmetros ao longo do tempo (HOLA; SCHABOWICKZ, 2010).

Um destes métodos é o ensaio de ultrassom, em que se utilizam transdutores que emitem e recebem pulsos ultrassônicos (de altas frequências) em elementos estruturais, medindo o tempo de viagem destes pulsos. O ensaio ainda pode ser aliado à técnica de tomografia computadorizada. Neste caso, são feitas várias medições de tempo em pontos diferentes da estrutura. Após a montagem e a resolução de um sistema de equações à partir destas medições, calculam-se as velocidades do pulso ultrassônico e a sua distribuição em uma seção transversal da estrutura, conhecida como tomograma. As regiões de baixa velocidade na seção representam heterogeneidades do material (trincas, defeitos de fabricação e/ou danos).

No processo de geração de imagens, é comum supor-se que a trajetória seguida pelos pulsos é retilínea. Dessa forma, os pulsos que atravessam estas regiões de heterogeneidade são mais lentos, visto que a velocidade do pulso se correlaciona com as propriedades mecânicas do material.

No entanto, a suposição de trajetórias retilíneas é distante do comportamento real dos pulsos, que desviam de regiões de baixa velocidade, buscando o caminho mais rápido possível entre os transdutores do ensaio de ultrassom. Neste trabalho, essa hipótese será abandonada, buscando mapear a trajetória percorrida pelos pulsos ultrassônicos em estruturas.

1.1 OBJETIVOS

O presente trabalho tem como objetivo definir a trajetória não linear de pulsos ultrassônicos em estruturas de concreto. Essa tarefa será desempenhada em seções transversais com distribuição de velocidades conhecida. Para isso, serão utilizados algoritmos determinísticos e bioinspirados. Anteriormente, estas trajetórias eram supostas retilíneas, simplificação distante do comportamento real. As alternativas determinísticas são os algoritmos de Dijkstra e A* (lê-se “A-estrela”), consagrados na resolução de problemas de *pathfinding*. As alternativas bioinspiradas são os algoritmos genéticos (GAs) e a otimização por colônia de formigas (ACO).

Dado o objetivo principal, traçam-se os seguintes objetivos específicos para atingi-lo:

a) Estudar os algoritmos determinísticos de Dijkstra e A*;
b) Estudar os algoritmos genéticos e de otimização por colônia de formigas;
c) Escolher os modelos de algoritmos bioinspirados que melhor se adequam ao problema de determinação da trajetória ótima, visto que estes algoritmos apresentam muitas variações de estrutura possíveis;
d) Implementar modelos escolhidos dos algoritmos no *software* TUSom, que gera imagens tomográficas a partir medições de tempo feitas com transdutores ultrassônicos. Isso é feito com o objetivo de obter as trajetórias ótimas do pulso ultrassônico em estruturas de concreto;
e) Avaliar a melhor alternativa para determinação das trajetórias ótimas: algoritmos determinísticos, bioinspirados ou abordagem mista, considerando-se dois aspectos principais: qualidade das respostas fornecidas e tempo de processamento. Como os algoritmos bioinspirados são probabilísticos e fornecem soluções diferentes a cada execução, não garantem a obtenção da solução ótima. Enquanto isso, as respostas obtidas pelos algoritmos

determinísticos são as mais precisas quanto possível e serão consideradas como referência para a qualidade das soluções obtidas;

f) Adotar o algoritmo que melhor alia os aspectos considerados como o padrão para a determinação das trajetórias no *software* TUSom.

1.2 JUSTIFICATIVA

Os ensaios não destrutivos vêm se difundindo cada vez mais, em especial o ultrassom. Quando se executam experimentos com o uso do ultrassom, obtêm-se os tempos de viagem dos pulsos. No entanto, as trajetórias destes pulsos ao longo da estrutura são incógnitas, visto que eles caminham pelos pontos de maior homogeneidade do material, que geram maior velocidade, de forma análoga à eletricidade, que caminha por onde há menor resistência. Estas trajetórias são supostas lineares, com a consciência de que esta é uma aproximação não coincidente com a realidade. Dessa forma, quando se realizam estes ensaios, resultados próximos do esperado são obtidos, mas o fenômeno não é completamente compreendido.

Quando a hipótese anterior é abandonada e trajetórias mais próximas das reais são obtidas, isso é feito de forma pouco eficaz através de algoritmos determinísticos, que testam todas as possibilidades de viagem do pulso dentro de uma malha discretizada no elemento de concreto, exigindo grande esforço computacional. Os algoritmos bioinspirados, neste trabalho representados pelos algoritmos genéticos e pela otimização por colônia de formigas, são técnicas estocásticas baseadas em processos da natureza, que têm se mostrado muito eficazes na resolução de problemas em diversos ramos da ciência, surgindo como uma alternativa viável e menos custosa computacionalmente à solução do problema das trajetórias. Em poucas palavras, sacrifica-se a qualidade da resposta (até um limite determinado e admissível) em prol de um tempo de processamento menor.

Resolver o problema das trajetórias significa possibilitar a melhoria do processo de geração de imagens tomográficas no futuro. Isso pode ser feito de algumas formas. Uma delas seria a seguinte: com as trajetórias ajustadas, é possível simular quais seriam as medições experimentais de tempo obtidas num ensaio de ultrassom a partir de um campo de velocidades predefinido. Usando estes valores de tempo, é possível gerar uma nova imagem, utilizando os processos já conhecidos ou novos. Dentre os conhecidos e já aplicados, existem o Método dos Mínimos Quadrados ou *Ordinary*

Least Squares (OLS), a Técnica de Reconstrução Algébrica ou *Algebraic Reconstruction Technique* (ART) e a Técnica de Reconstrução Iterativa Simultânea ou *Simultaneous Iteractive Reconstruction Technique* (SIRT). Dentre as novas técnicas, as Redes Neurais Artificiais (RNAs) se apresentam como uma opção. Comparando a imagem gerada com a imagem inicial, seria possível propor aprimoramentos ao processo de geração de imagens, comparar os conteúdos dessas imagens e tirar outras conclusões à respeito das técnicas utilizadas e da possibilidade de adotar novas técnicas.

1.3 CONTEÚDO DO TRABALHO

O conteúdo dos capítulos deste trabalho é sucintamente descrito abaixo:

a) No capítulo 2, “Revisão bibliográfica”, são esclarecidos os conceitos de ensaios não destrutivos, da tomografia computadorizada, de conceitos de ciência da computação e dos algoritmos determinísticos (Dijkstra e A*) e bioinspirados (algoritmos genéticos e otimização por colônia de formigas), trazendo o princípio de funcionamento de cada um. O funcionamento do *software* TUSom, de geração de imagens tomográficas em estruturas, também será explorado, incluindo algumas das formulações matemáticas utilizadas em sua elaboração;

b) No capítulo 3, “Metodologia”, a metodologia adotada no trabalho é exposta, detalhando as etapas de estudo e forma de implementação dos algoritmos determinísticos e bioinspirados no *software* TUSom, quais foram os valores utilizados para os parâmetros de cada um, quais exemplos foram escolhidos para teste e um resumo das etapas desenvolvidas no trabalho;

c) No capítulo 4, “Resultados e discussão”, mostra-se quais foram os resultados obtidos após o mapeamento das trajetórias do pulso ultrassônico nos exemplos de seções de concreto criados à partir de figuras, tabelas e gráficos, discutindo se os resultados obtidos eram esperados e possíveis motivos para sua obtenção, em conjunto com outras análises iniciais;

d) No capítulo 5, “Conclusões”, avaliam-se os resultados obtidos, constatando se o objetivo do trabalho foi atingido, ou seja, se o mapeamento do pulso foi bem-sucedido e qual(is) dos algoritmos apresentou o melhor desempenho nesta tarefa, tomando como referência os exemplos simulados e os parâmetros adotados para cada algoritmo. Além disso, os motivos deste desempenho superior foram explorados;

e) No capítulo 6, “Sugestões para trabalhos futuros”, esboçam-se algumas ideias do que pode ser feito ou aprofundado em próximos trabalhos, partindo das conclusões obtidas.

2 REVISÃO BIBLIOGRÁFICA

Nesta seção, serão tratados fundamentos teóricos necessários ao pleno entendimento e desenvolvimento deste trabalho. Inicialmente, os ensaios não destrutivos (ENDs) serão explorados, dando um enfoque maior para o ensaio de ultrassom, objeto de estudo da dissertação. Em seguida, trata-se da tomografia computadorizada, que pode ser aliada ao ensaio de ultrassom para gerar imagens de campos de velocidade do pulso ultrassônico. O *software* TUSom, usado para simular campos de velocidade de seções de estruturas e determinar a trajetória dos pulsos, também terá seu funcionamento investigado. Em sequência, conceitos teóricos de computação e programação são explorados, englobando os algoritmos aplicados na resolução do problema das trajetórias (determinísticos e bioinspirados). Por fim, faz-se um resumo dos assuntos tratados no capítulo.

2.1 ENSAIOS NÃO DESTRUTIVOS

A vida útil de uma estrutura de concreto armado é de cerca de 50 anos (MEHTA; MONTEIRO, 2014). A explosão desse sistema construtivo no Brasil se deu ao longo do século XX, especialmente nos anos 1940, motivada pela inserção de cimenteiras internacionais no país e por fatores como desempenho estrutural e facilidade operacional (SANTOS, 2008), segundo Souza e Ripper (1998, p. 4):

> ...as estruturas de concreto existentes estão envelhecendo, muitas já estão com dezenas de anos, os problemas de deterioração estão cada vez mais acentuados, exigindo com frequência trabalhos de recuperação e de reforço estrutural e mesmo, em casos mais graves, sua demolição.

Somada à situações de alteração funcional de estruturas, essa conjuntura tem motivado investimentos cada vez maiores em métodos e ensaios não destrutivos (ENDs) que sejam capazes de detectar propriedades, defeitos e o grau de deterioração de estruturas de forma não invasiva. Também constituem uma ferramenta importante no que diz respeito ao planejamento de ações de manutenção preventiva e ao monitoramento da integridade de estruturas de maior porte, como pontes e barragens, tendo em vista a garantia permanente de sua segurança. Em

alguns casos, este tipo de ensaio ainda constitui uma alternativa mais rápida que ensaios destrutivos convencionais (RAMÍREZ, 2015).

Essas tecnologias vêm evoluindo lentamente devido à estrutura heterogênea de materiais como o concreto e a madeira. Tratando especificamente do concreto, existem muitos tipos de ensaios não destrutivos (MEHTA; MONTEIRO, 2014):

a) Ensaio de resistência à penetração;
b) Ensaio de arrancamento;
c) Método de dureza superficial;
d) Teste de maturidade;
e) Ensaios de medição de absorção e permeabilidade;
f) Métodos de propagação de ondas de tensão;
g) Métodos elétricos, eletromagnéticos e eletroquímicos;
h) Ensaio de avaliação de frequências de ressonância;
i) Métodos radioativos/nucleares;
j) Técnicas de termografia infravermelha;
k) Método de emissão acústica.

Dentre os métodos, destacam-se os métodos de propagação de ondas de tensão, grupo do qual o ensaio de ultrassom faz parte.

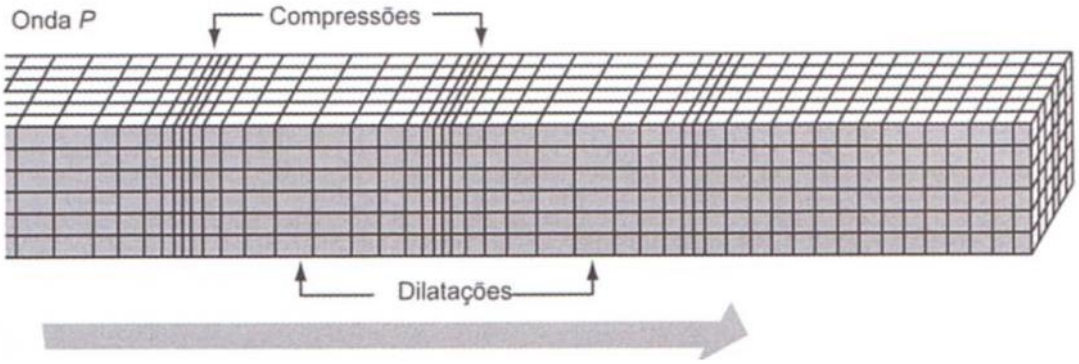
2.2 MÉTODOS DE PROPAGAÇÃO DE ONDAS DE TENSÃO

Antes de explorar os métodos de propagação de ondas de tensão, é necessário compreender alguns conceitos básicos de propagação de ondas em meios sólidos. Existem quatro tipos de ondas principais que se propagam nesse tipo de meio:

a) Ondas longitudinais:

Este tipo de onda se caracteriza pelo movimento das partículas ocorrendo na mesma direção da propagação de onda (Figura 2.1), causando alteração no volume do sólido (MEHTA; MONTEIRO, 2014). Se propagam de forma análoga à ondas sonoras no ar (MALHOTRA; CARINO, 2004).

Figura 2.1 - Ondas longitudinais (ou ondas P) se propagando em um sólido.



Fonte: Mehta e Monteiro, 2014.

Também são chamadas de ondas primárias (ondas P), pois possuem velocidades maiores que os outros tipos de ondas, sendo as primeiras a serem detectadas numa ocasião de terremoto, por exemplo, problema estudado pelos geofísicos e que muito contribuiu para a compreensão dos fenômenos de propagação de ondas (MEHTA; MONTEIRO, 2014).

Para um material homogêneo e isotrópico, a velocidade de propagação de ondas longitudinais pode ser descrita em função do módulo de Young (E), do coeficiente de Poisson (ν) e da massa específica do material (ρ), vide equação 2.1.

$$V_p = \sqrt{\frac{E(1-\nu)}{\rho(1+\nu)(1-2\nu)}} \tag{2.1}$$

Para o concreto, essas velocidades variam entre 3000 e 5000 m/s na maior parte dos casos (MALHOTRA; CARINO, 2004). Whitehurst[1] (1966, apud RAMIREZ, 2016, p. 62) elaborou uma tabela (Tabela 2.1) com velocidades de propagação de pulsos ultrassônicos em concreto, relacionando-as com a qualidade do mesmo, que pode servir como ponto de partida. No entanto, o autor recomenda uma análise caso a caso para melhores resultados, avaliando a velocidade de propagação no concreto considerado de boa qualidade e tomando esta como referência para comparações.

[1]WHITEHURST, E. A. **Evaluation of concrete properties from sonic test.** Ann Arbor: American Concrete Institute, 1966.

Tabela 2.1 - Correlação entre a velocidade das ondas P e a qualidade do concreto.

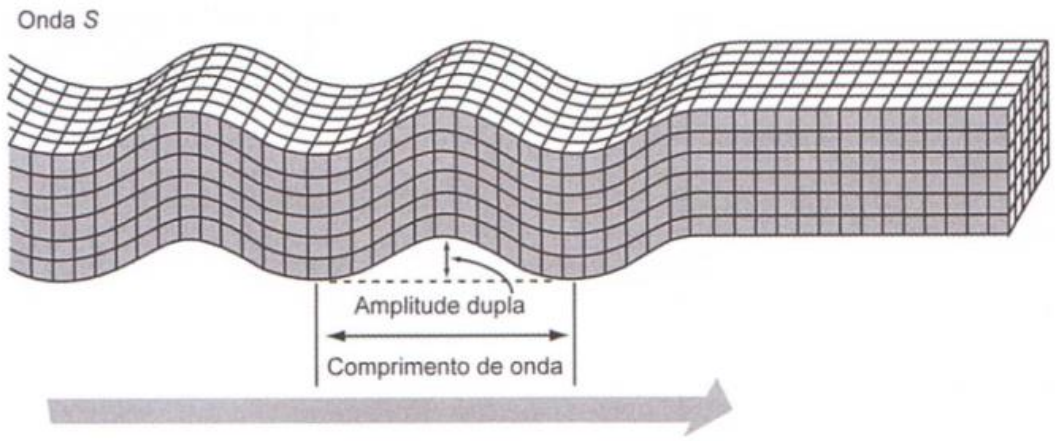
V (m/s)	Qualidade
> 4575	Excelente
3660 - 4575	Boa
3050 - 3660	Regular
2135 - 3050	Ruim
< 2135	Muito ruim

Fonte: Whitehurst, 1966.

b) Ondas transversais:

Se caracterizam pelo movimento das partículas do sólido na direção transversal à da propagação de onda, não causando alteração em seu volume (Figura 2.2) (MEHTA; MONTEIRO, 2014).

Figura 2.2 - Ondas transversais (ou ondas S) se propagando em um sólido.



Fonte: Mehta e Monteiro, 2014.

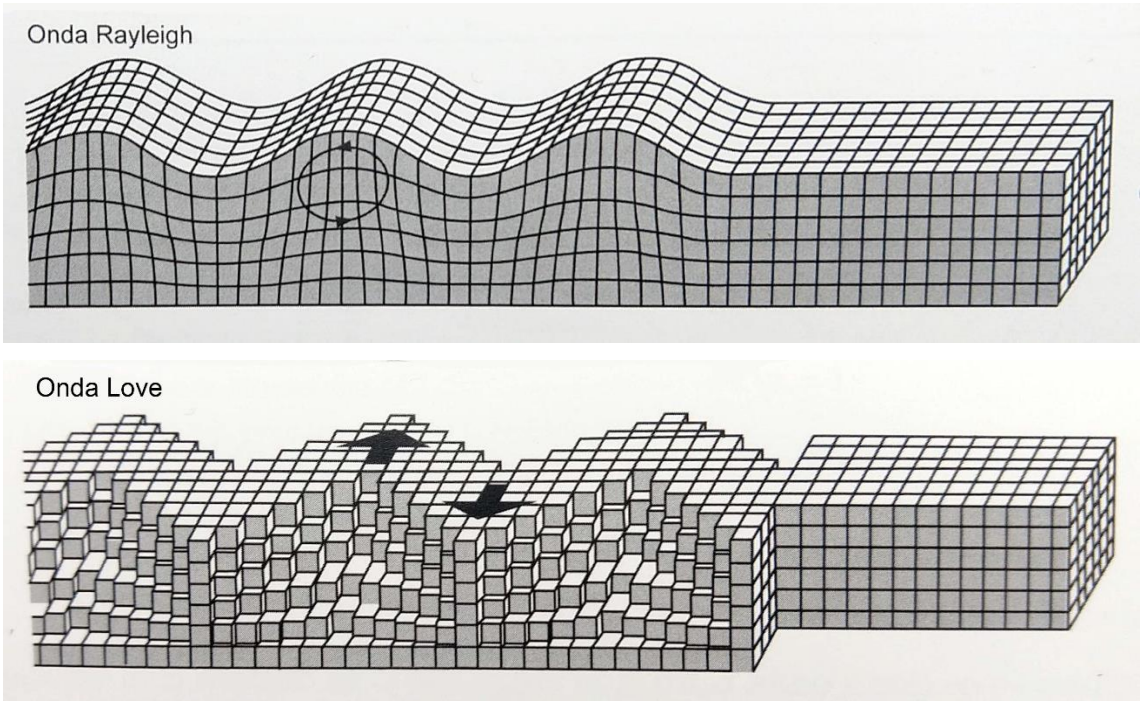
Também são chamadas de ondas secundárias (ondas S), pois possuem velocidade da ordem de 60% da velocidade das ondas longitudinais (são mais lentas e portanto chegam depois). Essa velocidade pode ser descrita de maneira análoga ao que foi feito para as ondas longitudinais (material homogêneo e isotrópico), vide equação 2.2.

$$V_s = \sqrt{\frac{E}{2\rho(1+\nu)}} \tag{2.2}$$

c) Ondas Rayleigh e ondas Love (ondas superficiais):

Estão presentes junto à superfície do material. A primeira pode ser vista como a combinação das ondas P e S, onde as partículas vibram em um movimento elíptico (Figura 2.3). Enquanto isso, a segunda se caracteriza pelo movimento das partículas num plano horizontal perpendicular à direção de propagação da onda, ou seja, de forma semelhante ao que ocorre nas ondas transversais, mas em outra direção (MEHTA; MONTEIRO, 2014). As ondas superficiais são as mais lentas dentre as apresentadas, com uma velocidade da ordem de 55% da velocidade das ondas longitudinais (MALHOTRA; CARINO, 2004).

Figura 2.3 - Ondas Rayleigh e ondas Love se propagando em um sólido.



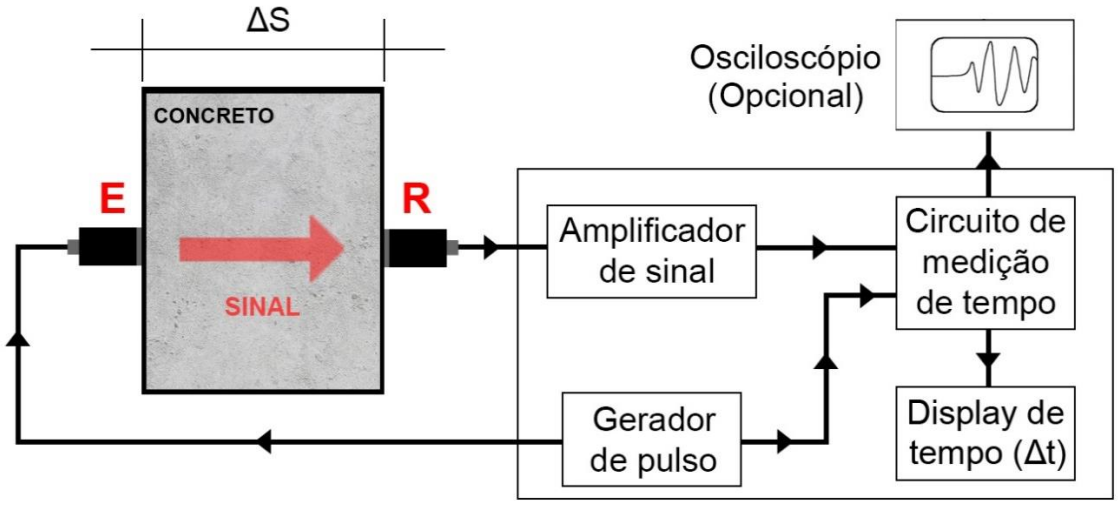
Fonte: Mehta e Monteiro, 2014.

Segundo Mehta e Monteiro (2014, p. 434): “O modo como uma onda se reflete e refrata através de um material sólido pode fornecer informação vital sobre sua heterogeneidade interna”. Dentre os ensaios não destrutivos do concreto, no grupo de métodos de propagação de ondas de tensão, os principais são o método do eco-impacto e do ultrassom (se subdivide nas modalidades pulso-eco e transparência). Estes métodos têm em comum a característica de medir os tempos de viagem e inferir propriedades e heterogeneidades do concreto à partir do efeito que a estrutura gera

na propagação de ondas de tensão emitidas. Eles diferem em pontos como princípio de funcionamento, instrumentação e técnicas de processamento de sinais utilizadas (MALHOTRA; CARINO, 2004).

O ensaio de ultrassom, como o próprio nome sugere, consiste na emissão e recepção de pulsos ultrassônicos (frequências de impacto acima de 20kHz) por transdutores piezoelétricos. Quando a emissão e a recepção do sinal são feitas por um mesmo transdutor, detectando a reflexão do pulso, tem-se a modalidade pulso-eco. Quando um transdutor emite o sinal e outro recebe, tem-se a modalidade transparência, cujo arranjo se encontra representado na Figura 2.4 (MALHOTRA; CARINO, 2004).

Figura 2.4 - Arranjo de ensaio de ultrassom na modalidade transparência.



NOTA: E – Transdutor emissor; R – Transdutor receptor.

Fonte: Adaptado de Malhotra e Carino, 2004.

O foco do presente trabalho é simular computacionalmente o ensaio de ultrassom na modalidade transparência. Além da detecção de heterogeneidades no concreto, este pode ser utilizado para diversas outras finalidades: caracterizar suas propriedades físicas e mecânicas, estudar sua durabilidade e processos de hidratação e medir a profundidade e o tamanho de fissuras. Também é possível estimar o módulo de elasticidade dinâmico do concreto (MALHOTRA; CARINO, 2004).

Os pulsos ultrassônicos emitidos pelos transdutores possuem parâmetros como amplitude, frequência e comprimento de onda, que sofrem variações ao passar por zonas heterogêneas do material, permitindo a detecção de defeitos à partir da

identificação de regiões de alta ou baixa velocidade e/ou da atenuação da amplitude da onda inicialmente emitida (RAMÍREZ, 2015). Quanto mais uniforme e menos poroso for o sólido, maior é a velocidade de propagação dos pulsos ultrassônicos em seu interior.

O equipamento utilizado no Laboratório do Departamento de Engenharia de Estruturas da EESC-USP para a realização do ensaio de ultrassom encontra-se representado na Figura 2.5. Este mede o tempo de chegada do primeiro pulso ultrassônico entre dois transdutores, sendo portanto o tempo de viagem das ondas longitudinais.

Figura 2.5 - Equipamento de Ultrassom Pundit Lab^{+}.

Fonte: PROCEQ SA (2017).

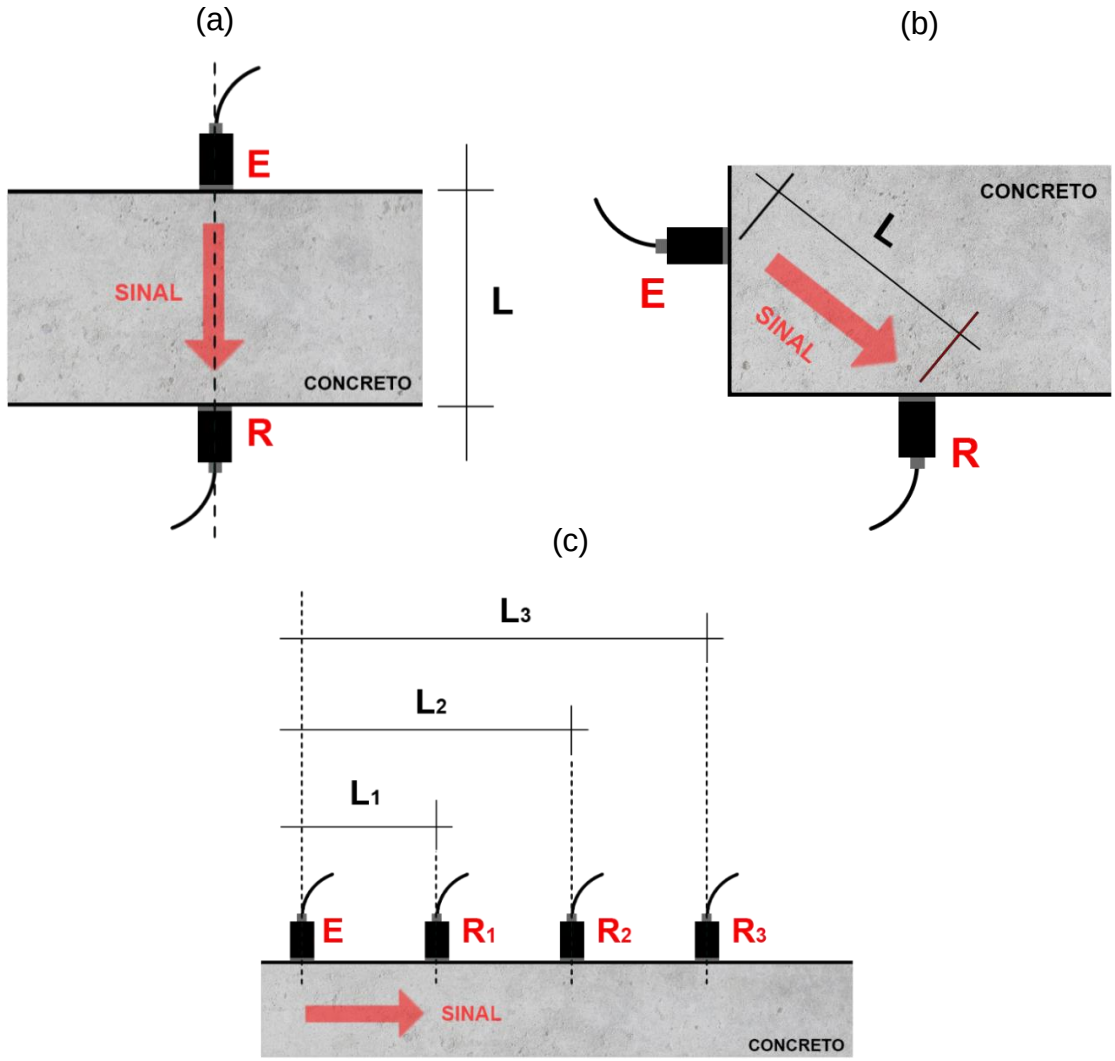
Os transdutores são alocados em pontos diferentes da estrutura, vide Figura 2.4. Para garantir que haja perfeita aderência entre os transdutores e o concreto, utiliza-se um gel acoplante, que pode ser gel médico, vaselina, graxa, mel e outros. Através de um material piezoelétrico, que converte sinais elétricos em deformação, o transdutor emissor gera uma onda mecânica (HAACH, 2017). Calcula-se a velocidade de propagação desta onda através da equação 2.3, onde V representa a velocidade (m/s), ΔS representa a distância entre os transdutores (m), conhecida à partir do arranjo experimental, e Δt representa o tempo de viagem da onda (s), grandeza medida pelos transdutores.

$$V = \frac{\Delta S}{\Delta t} \tag{2.3}$$

Quando os transdutores emissor e receptor se encontram em faces opostas do elemento estrutural, temos o arranjo direto (Figura 2.6.a). Ainda é chamado de ortogonal se os transdutores estiverem alinhados, formando ângulos de 90º com a direção oposta à sua alocação. O arranjo é chamado oblíquo quando o anterior não ocorre. Este tipo de medida é o mais recomendado quando possível, pois faz com que o sinal seja recebido com maior intensidade (ABNT, 2019).

Outro arranjo possível é o semidireto (Figura 2.6.b), ao qual se recorre quando não é possível alocar os transdutores em faces opostas. Esse arranjo também é adotado para evitar regiões com grande densidade de armadura, que interferem nas medições (MALHOTRA; CARINO, 2004).

Figura 2.6 - Arranjo de transdutores no ensaio de ultrassom (modalidade transparência).



a) Direto; b) Semidireto; c) Indireto.

NOTA: E – Transdutor emissor; R – Transdutor receptor.

Fonte: Adaptado de ABNT (2019).

Por fim, tem-se o arranjo indireto (Figura 2.6.c), utilizado quando os outros não são possíveis (uma das faces da estrutura não é acessível). Este consiste na disposição dos transdutores emissor e receptor na mesma face da estrutura, quando o comprimento de uma face permite a movimentação do transdutor receptor, pois este é deslocado para que se realizem diversas medidas de tempo (ABNT, 2019). É o arranjo menos desejável, pois atenua em grande parte a intensidade do sinal recebido pelo transdutor receptor e está mais sujeito a erros (MALHOTRA; CARINO, 2004).

Além do arranjo dos transdutores, outras variáveis contribuem para a alteração da velocidade de propagação dos pulsos ultrassônicos: idade do concreto, tamanho e formato da estrutura, condições de umidade, quantidade e tipo do agregado, existência de microfissurações e presença de armadura (MEHTA; MONTEIRO, 2014).

Apesar da medição não apresentar grandes dificuldades, a representação convencional dos resultados do ensaio de ultrassom é deficiente, pois tenta representar o que ocorre em uma seção bidimensional através de um gráfico unidimensional. Dessa forma, informações valiosas contidas na seção se perdem (PERLIN; PINTO, 2013). A visualização dos resultados do ensaio torna-se então um desafio, solucionado pelo uso da técnica de tomografia computadorizada.

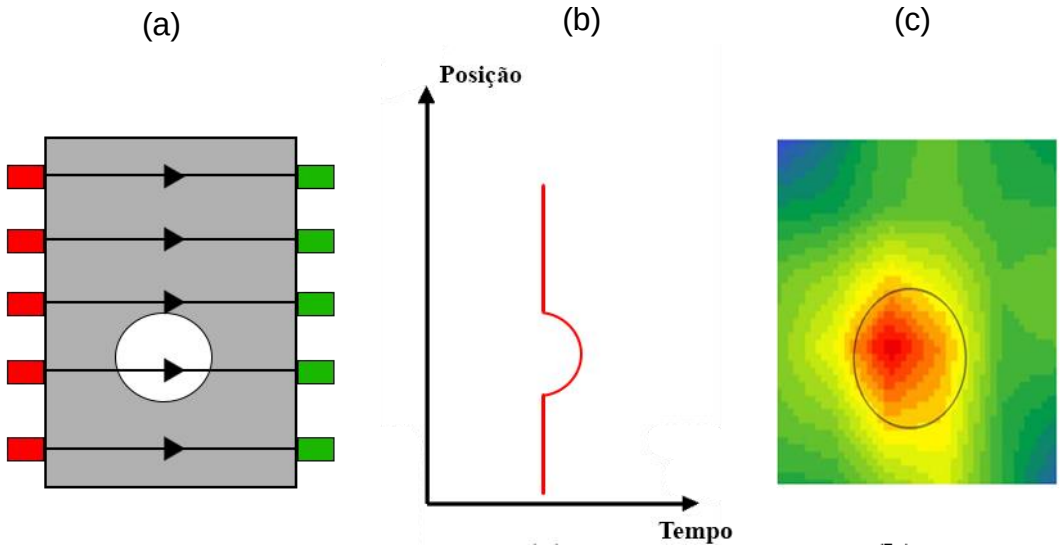
2.3 TOMOGRAFIA COMPUTADORIZADA (TC)

Em 1917, o matemático austríaco Johann Radon provou ser possível a reconstrução completa de qualquer objeto tridimensional à partir de múltiplas projeções do mesmo, feitas de ângulos diferentes (DEANS, 1983). Apesar da descoberta no início do século, os cálculos eram extremamente complexos e trabalhosos, o que fez com que o primeiro equipamento de TC fosse inventado somente em 1972 (FILLER, 2009), se aproveitando do desenvolvimento acelerado das ferramentas computacionais na época. Desde então, o processo se popularizou no meio médico, sendo muito comum nos dias atuais.

Quando aplicado ao ensaio de ultrassom, a TC permite representar a distribuição das velocidades do pulso ultrassônico em seções transversais de estruturas. Para isso, são necessárias múltiplas medições de velocidade em diferentes pontos da seção e a aplicação de algoritmos de reconstrução de imagens. A Figura 2.7 compara a representação de uma heterogeneidade no concreto pelo

método unidimensional convencional e pelo método bidimensional com o uso da tomografia (em escala de cores contínua), chamada tomograma.

Figura 2.7 - Representação de um defeito no ultrassom.

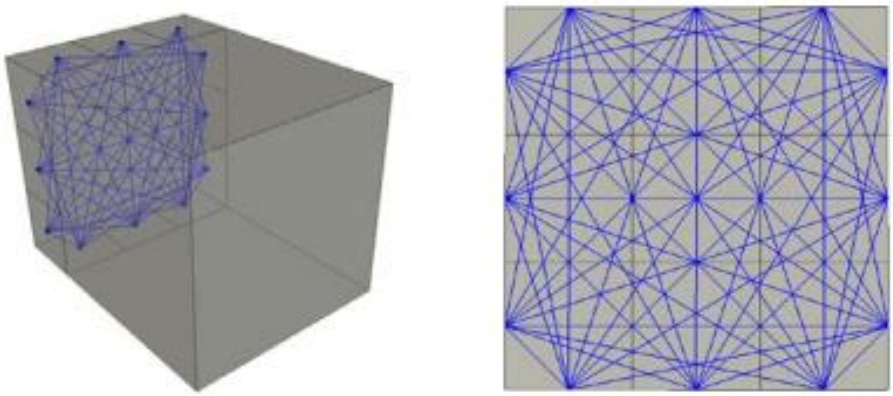


a) Seção de concreto com defeito e transdutores de ultrassom;
b) unidimensional convencional; c) bidimensional com tomografia (tomograma).

Fonte: Adaptado de Ramírez (2015).

Ramírez (2015) determina o arranjo de transdutores mais vantajoso do ponto de vista do número de medições realizadas em uma seção pela qualidade da imagem gerada com a tomografia computadorizada, mostrado na Figura 2.8. Neste esquema, são realizados praticamente todos os tipos de medição: diretas ortogonais, diretas oblíquas e semidiretas.

Figura 2.8 - Esquema de medições de ultrassom em uma seção transversal de concreto.



Fonte: Ramírez (2015).

Existem dois caminhos principais adotados para a geração de imagens tomográficas: a retroprojeção filtrada, que geralmente envolve o processo da transformada de Radon, e a reconstrução iterativa, que se baseia na resolução algébrica de um sistema de equações (HAACH, 2017). Como estes sistemas apresentam grandes dimensões, sua resolução costuma ser computacionalmente custosa (PERLIN, 2011).

2.4 SOFTWARE TUSOM

O *software* TUSom foi desenvolvido pelo Prof. Dr. Vladimir Guilherme Haach em linguagem de programação Pascal, no ambiente de desenvolvimento Lazarus, com a colaboração de alunos de mestrado do Departamento de Engenharia de Estruturas (SET) da EESC-USP.

O TUSom tem o intuito de mapear as velocidades do pulso ultrassônico em seções de uma estrutura à partir de medidas experimentais de tempo fornecidas pelo ensaio de ultrassom. O resultado final é a geração de tomogramas como o da Figura 2.7. A aplicação do *software* é feita em seis etapas principais:

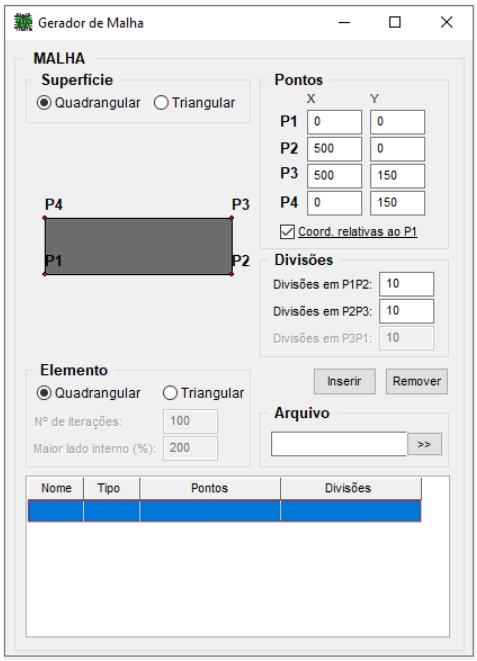
a) Entrada da geometria da seção transversal e do tamanho da malha de discretização que será adotada em laboratório (depende do diâmetro dos transdutores utilizados), semelhante ao que ocorre no método dos elementos finitos e representada na Figura 2.9. Tanto a geometria da seção quanto os elementos da malha podem assumir formato quadrangular ou triangular;
b) Entrada das coordenadas dos pontos de medição das trajetórias, que representam os pontos onde os transdutores serão posicionados na seção do corpo de prova. Estes pontos são considerados como os pontos médios das arestas do contorno exterior da malha, como mostrado na Figura 2.10;
c) Definição das linhas de medição das trajetórias (Figura 2.11):

As trajetórias foram inicialmente supostas como retilíneas, hipótese adotada em Kwon, Choi e Song (2005), Yanli (2010), Chai *et al.* (2010 e 2011), Aggelis *et al.* (2011), Perlin (2011), Ferraro, Boyd e Consolazio (2013), entre outros. Essa aproximação é razoável, mas não totalmente concordante com a realidade, visto que o pulso ultrassônico viaja pelos pontos de maior homogeneidade da estrutura. Para um caso em que existam defeitos/danos pronunciados, como o exemplo da Figura

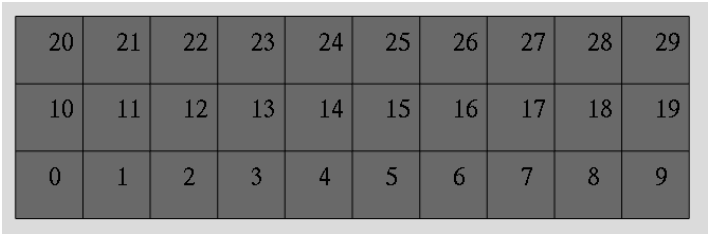
2.12, a suposição introduz erros ao cálculo dos tempos de viagem e consequentemente à imagem gerada como resultado, diminuindo a sua qualidade.

Figura 2.9 - Etapa de entrada de geometria no *software* TUSom.

(a)



(b)

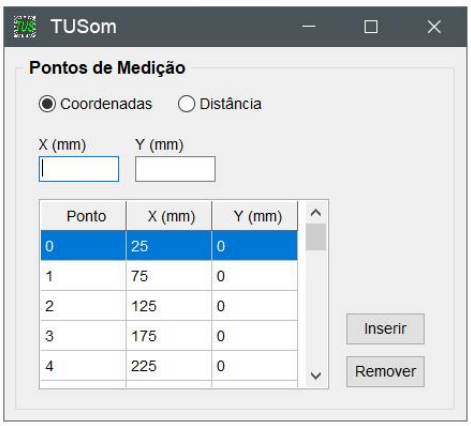


a) Geometria; b) Malha escolhida com elementos numerados.

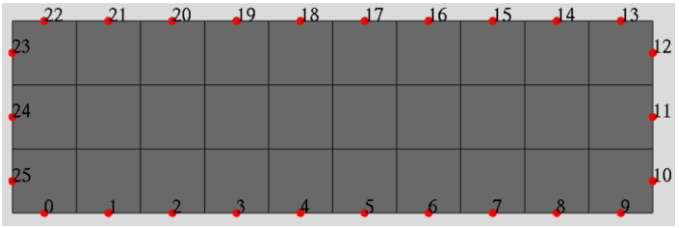
Fonte: Autor.

Figura 2.10 – Etapa de definição dos pontos de medição no *software* TUSom.

(a)



(b)

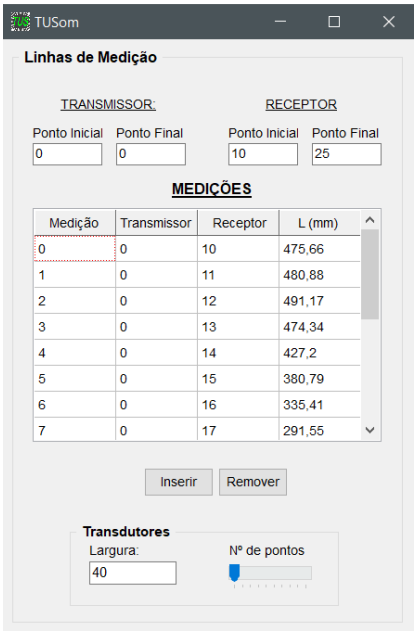


a) Coordenadas dos pontos de medição; b) Pontos numerados na malha.

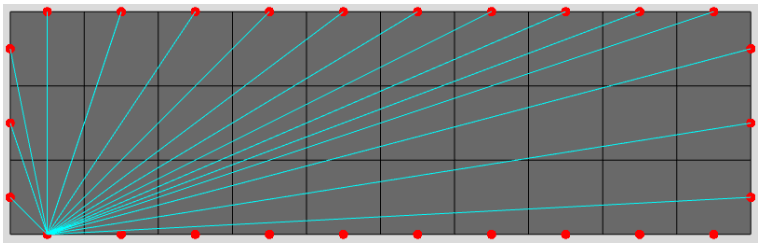
Fonte: Autor.

Figura 2.11 – Definição das linhas de medição das trajetórias no *software* TUSom.

(a)



(b)

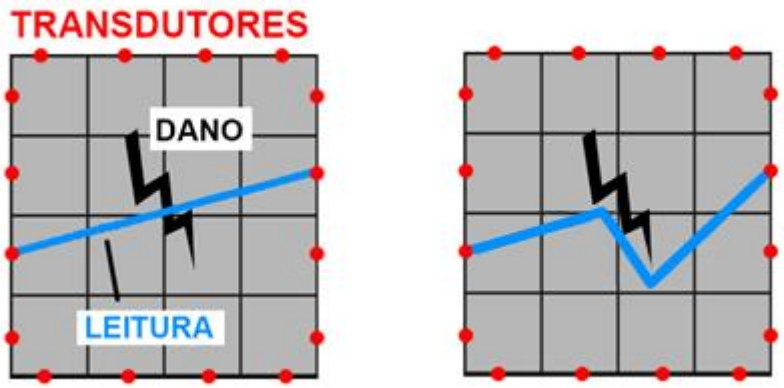


a) Definição das linhas de medição; b) Linhas de medição traçadas na malha.

Fonte: Autor.

Figura 2.12 - Comparação entre trajetórias numa seção com defeito pronunciado.

(a) (b)



NOTA: a) Trajetória suposta; b) Trajetória real.

Fonte: Autor.

Segundo Jackson e Tweeton (1994), quando a diferença das velocidades apresentadas no meio não é maior que 10%, a aproximação das trajetórias dos pulsos como retilíneas não resulta em grandes distorções e é quase sempre aceitável. Para valores acima desta faixa, o resultado pode ser aceitável dependendo do usuário, mas recomenda-se a adoção de estratégias que mapeiem o trajeto do pulso ao longo da seção experimentada.

d) Entrada de dados de medições experimentais (tempos de viagem), representada na Figura 2.13. Para cada medição, são efetuadas várias leituras de tempo (número definido pelo operador do equipamento do ensaio) e o *software* calcula parâmetros como média, desvio padrão e coeficiente de variação, permitindo o descarte de medidas pouco representativas;

Figura 2.13 - Entrada das medições de tempo experimentais no *software* TUSom.

Fonte: Autor.

e) Cálculo das parcelas da trajetória que os pulsos irão seguir em cada um dos elementos da malha:

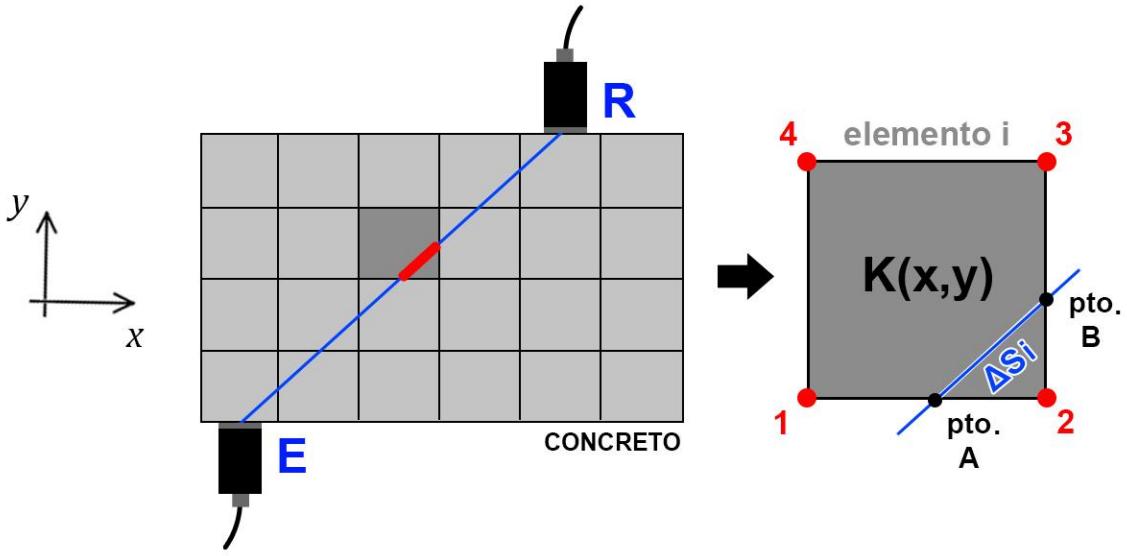
O tempo total de viagem do pulso é considerado como a somatória dos tempos de viagem em cada um dos elementos da malha discretizada, em um desenvolvimento análogo ao da equação 2.3. Define-se agora uma nova grandeza, chamada vagarosidade, cujo valor é o inverso da velocidade e que é representada pela letra K, vide equação 2.4.

$$K = \frac{1}{V} \tag{2.4}$$

Essa grandeza é calculada para cada um dos nós de um elemento (situados em suas extremidades). Esse processo é baseado na formulação exposta por Haach (2017) e exemplificado na Figura 2.14 para o caso de discretização da seção em elementos quadrangulares (4 nós). Supõe-se que a interpolação do campo de

vagarosidades (K) é linear ao longo do elemento, de forma semelhante ao que ocorre com os deslocamentos quando se utiliza o método dos elementos finitos.

Figura 2.14 – Cálculo das parcelas de uma trajetória percorridas em cada elemento.



Fonte: Adaptado de Haach (2017).

O campo de vagarosidades é escrito na forma do polinômio da equação 2.5, função das coordenadas x e y do elemento quadrangular:

$$K(x,y) = \alpha_1.1 + \alpha_2.x + \alpha_3.y + \alpha_4.x.y \tag{2.5}$$

A equação 2.5 pode ser escrita na forma matricial, conforme segue:

$$K(x,y) = \{1 \quad x \quad y \quad x.y\} \begin{Bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{Bmatrix} = \{p(x,y)\}^T \{\alpha\} \tag{2.6}$$

Os parâmetros α são definidos à partir dos valores de vagarosidade nodais, de acordo com a equação 2.7.

$$\begin{Bmatrix} K_1 \\ K_2 \\ K_3 \\ K_4 \end{Bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1.y_1 \\ 1 & x_2 & y_2 & x_2.y_2 \\ 1 & x_3 & y_3 & x_3.y_3 \\ 1 & x_4 & y_4 & x_4.y_4 \end{bmatrix} \begin{Bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{Bmatrix} \rightarrow \{K\} = [A]\{\alpha\} \rightarrow \{\alpha\} = [A]^{-1}\{K\} \tag{2.7}$$

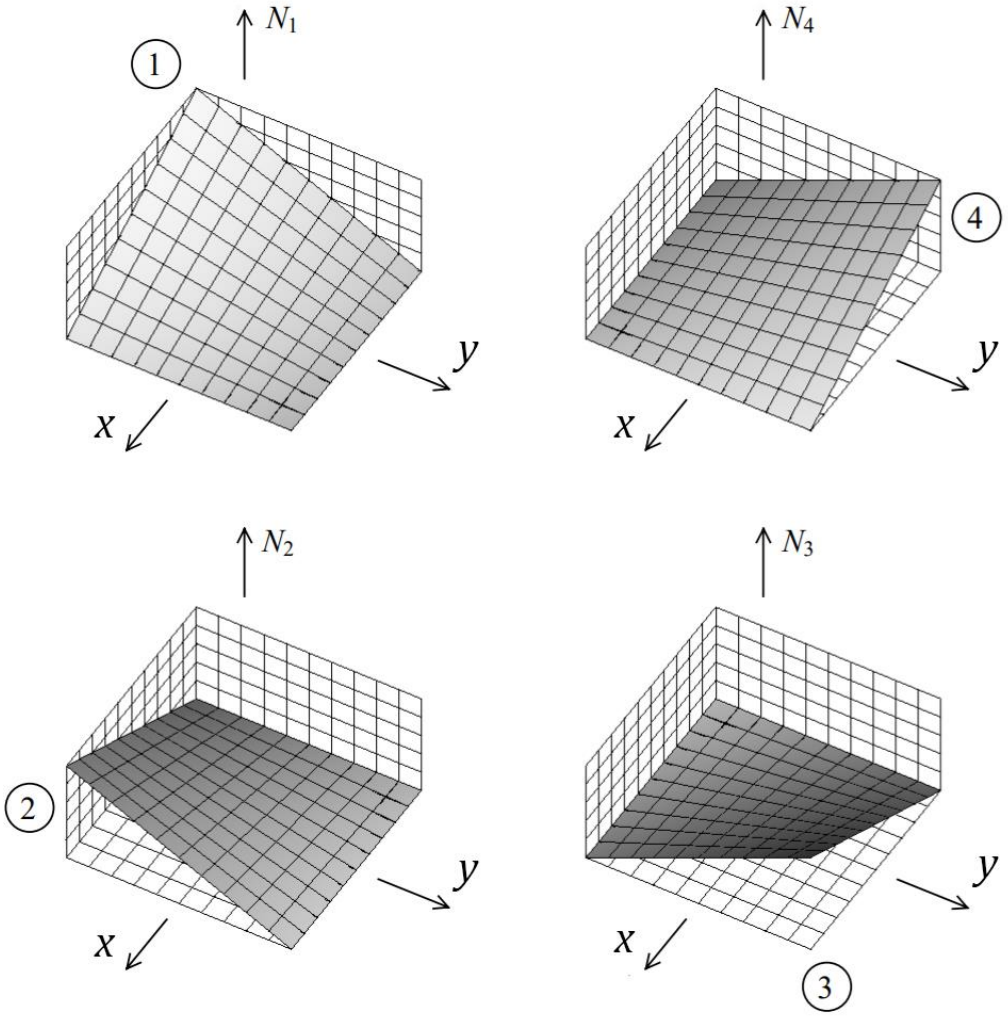
Substituindo a equação 2.7 na equação 2.6, tem-se o campo de vagarosidades escrito em função das vagarosidades nodais (equação 2.8). Surgem então quatro funções de forma (uma para cada nó), resultado da multiplicação do vetor $\{p(x,y)\}^T$ pela inversa da matriz A (matriz das coordenadas de cada um dos nós do elemento).

$$K(x,y) = \{p(x,y)\}^T [A]^{-1}\{K\} \rightarrow K(x,y) = [N(x,y)]\{K\} \tag{2.8}$$

O vetor de funções de forma apresenta-se na equação 2.9. Estas funções têm a característica principal de assumir valor 1 no nó em questão e 0 nos outros nós do elemento, situação ilustrada pela Figura 2.15.

$$[N(x,y)] = [N_1(x,y) \quad N_2(x,y) \quad N_3(x,y) \quad N_4(x,y)] \tag{2.9}$$

Figura 2.15 – Gráfico das funções de forma para cada nó de um elemento quadrangular.



Fonte: Azevedo (2003).

Dessa forma, o tempo de viagem do pulso ultrassônico dentro do elemento i toma a forma da equação 2.10. Esta representa uma integral em linha, visto que as vagarosidades estão escritas à partir das funções de forma, que são polinômios definidos. Na equação, ΔS_i representa o trecho da trajetória de uma medida que atravessa o elemento i, calculado à partir da equação 2.11 para pontos A e B quaisquer, representados na Figura 2.14. A equação parte da aplicação do teorema de Pitágoras.

$$\Delta t_i = \int_0^{\Delta Si} K(x,y)\, dS \tag{2.10}$$

$$\Delta S_i = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \tag{2.11}$$

Aplicando a equação 2.10 a todos os elementos da malha e efetuando a soma, tem-se o tempo total de uma medição. Quando determinada medição não atravessa algum elemento, Δt_i assume valor nulo neste trecho. Repetindo o procedimento para todas as medições, é possível saber quais elementos cada medição atravessou, a distância percorrida em cada um e qual foi o tempo gasto neste percurso, montando ao final do processo o vetor de tempos totais $\{T\}_m$ e a matriz de distâncias $[S]_{m,n}$, onde m representa o total de medições e n representa o número de elementos em que a malha foi discretizada. Tem-se então um sistema linear de equações, representado na forma matricial pela equação 2.12. O vetor K representa o vetor de vagarosidades em cada um dos elementos da malha (incógnita do problema).

$$\{T\}_m = [S]_{m,n}.\{K\}_n \tag{2.12}$$

O sistema da equação 2.12 apresenta características peculiares, pois sua determinação depende do número de medições realizado. Caso o número de medições (equações) seja maior que o número total de elementos nos quais a seção foi discretizada (incógnitas), ele é um sistema sobredeterminado (caso mais comum). Nesta situação, geralmente o sistema é inconsistente e não existe solução que satisfaça todas as equações simultaneamente. Em casos mais raros, o sistema é consistente e possui solução única. Caso o número de medições seja igual ao número

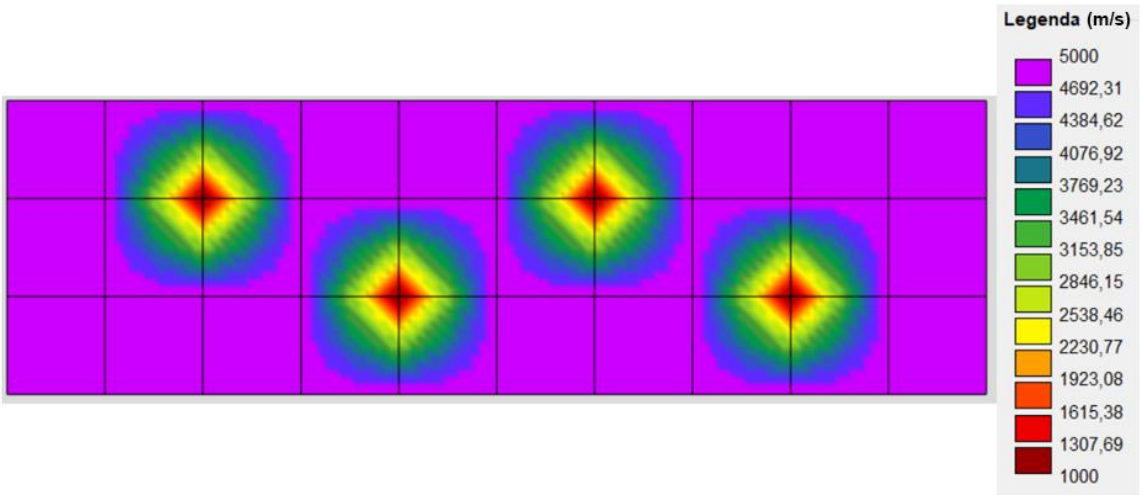
de elementos, tem-se um sistema determinado (mesmo número de equações e de incógnitas), com solução única. No último caso possível, existem menos equações do que incógnitas e o sistema é indeterminado, possuindo infinitas soluções (PERLIN; PINTO, 2019).

Além disso, erros experimentais são incorporados às equações, criando um sistema inconsistente, onde não existe solução única, qualquer que seja o número de medições. Dessa maneira, processos de inversão simples ou eliminação gaussiana não podem ser aplicados (PERLIN; PINTO, 2019) e é necessário recorrer à métodos mais complexos de resolução, apresentados a seguir.

f) Processo de reconstrução de imagens:

O resultado do sistema linear da equação 2.12 é o vetor de vagarosidades de cada elemento. Partindo deste, encontram-se as vagarosidades nodais à partir da equação 2.8. Com as vagarosidades nodais, calculam-se as velocidades nodais (uma grandeza é o inverso da outra, vide equação 2.4) e, através de um processo de interpolação linear destas velocidades, gera-se a imagem do campo de velocidades de uma seção transversal de concreto, em escala de cores contínua (Figura 2.16).

Figura 2.16 - Geração de imagens em escala de cores contínua no *software* TUSom.



Fonte: Autor.

É possível optar por três técnicas para a resolução do sistema da equação 2.12. A primeira delas é o método dos mínimos quadrados (*Ordinary Least Squares* – OLS), que consiste em minimizar a somatória do quadrado dos erros (resíduo), calculada à partir da diferença entre as medições experimentais e as medições teóricas de tempo

de viagem do pulso (HAACH, 2017). Os valores das medições experimentais são obtidos através do ensaio de ultrassom, enquanto as medições teóricas são obtidas à partir da formulação exposta na equação 2.10. A equação 2.13 apresenta o cálculo do resíduo, em que i representa cada elemento e j representa cada medição.

$$r^2 = \sum_j \left(\Delta t_{exp,j} - \Delta t_j\right)^2 = \sum_j \left(\Delta t_{exp,j} - \sum_i \int_0^{\Delta Si} K_i(x,y)\, dS\right)^2 \tag{2.13}$$

Para minimizar o resíduo, impomos a condição da equação 2.14, em que a segunda derivada do resíduo em relação às vagarosidades de cada elemento deve ser nula. À partir disso, temos uma estimativa teórica da vagarosidade em cada um dos elementos em função das medições realizadas experimentalmente (RAMÍREZ, 2015).

$$\frac{\partial r^2}{\partial K_i(x,y)} = 0 \tag{2.14}$$

A segunda opção para a resolução do sistema é a técnica de reconstrução algébrica (*Algebric Reconstruction Technique* - ART). Ela provém do método das projeções proposto por Karcmarz (1937) e consiste em propor um palpite inicial para o vetor de vagarosidades $\vec{K}$ e em seguida projetá-lo sobre o plano definido pela primeira equação do sistema linear. O vetor obtido é projetado sobre a segunda equação do sistema e assim consecutivamente, até a última equação. Após este processo, o vetor resultante é projetado novamente sobre a primeira equação e o desenvolvimento se repete (HAACH; JULIANI, 2014). Caso exista solução única para o sistema de equações, ela sempre será encontrada (KAK; SLANEY, 2001).

Para o exemplo em que sejam feitas duas medições de tempo em laboratório (equações do sistema) e a seção transversal seja dividida em dois elementos (incógnitas do sistema), o sistema linear da equação 2.12 passa a ser determinado e com dimensões 2x2, tomando a forma da equação 2.15.

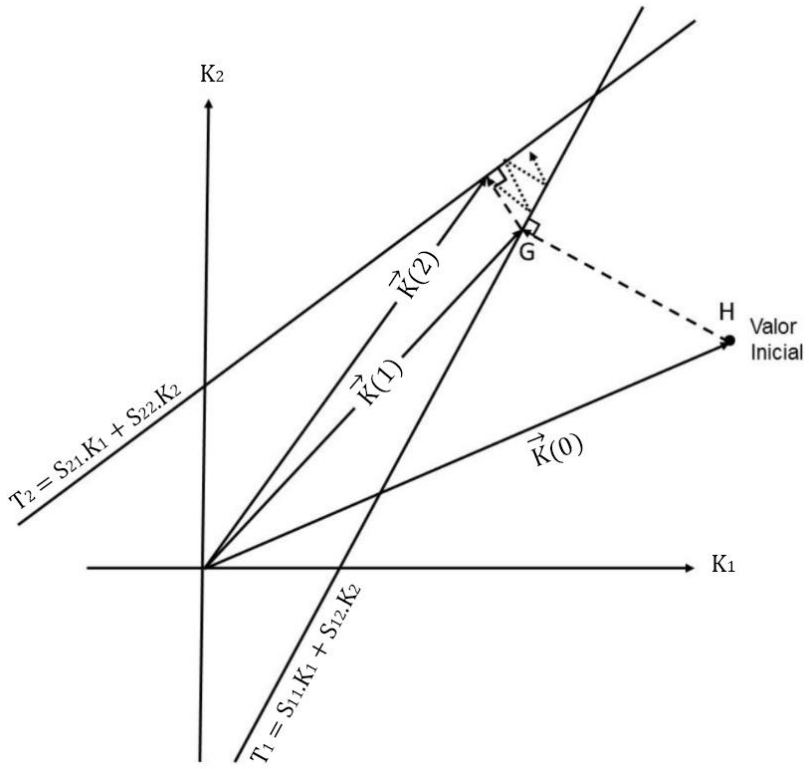
$$\begin{Bmatrix} T_1 \\ T_2 \end{Bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{Bmatrix} K_1 \\ K_2 \end{Bmatrix} \tag{2.15}$$

Desenvolvendo esse sistema, temos:

$$\begin{cases} T_1 = S_{11}.K_1 + S_{12}.K_2 \\ T_2 = S_{21}.K_1 + S_{22}.K_2 \end{cases} \quad (2.16)$$

É possível representar graficamente a aplicação do algoritmo para essa situação, já que temos apenas duas incógnitas, ou seja, as retas que representam cada uma das equações podem ser retratadas em um mesmo plano (Figura 2.17).

Figura 2.17 – Ilustração do método de resolução do ART para um sistema 2x2.

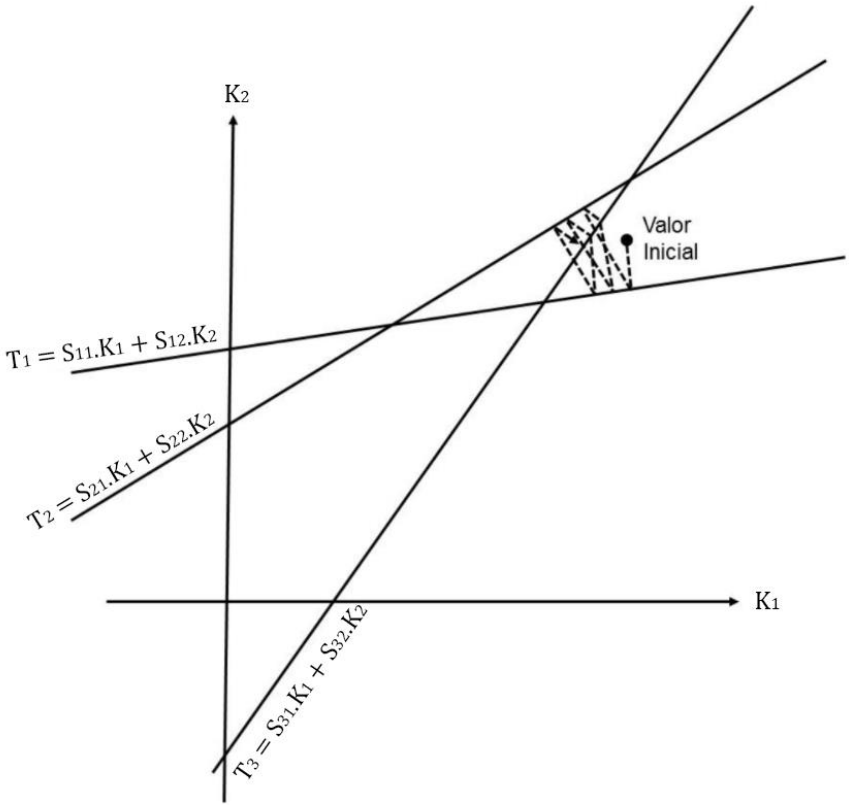


Fonte: Adaptado de Kak e Slaney (2001).

Para a situação em que forem feitas três medições em laboratório e a seção transversal for dividida em dois elementos, temos um sistema de dimensões 3x2 (três equações e duas incógnitas) e portanto sobredeterminado (mais medições que elementos, caso mais comum). Nessa situação, o desenvolvimento é análogo ao que foi feito nas equações 2.15 e 2.16 e temos como resultado três equações, que também podem ser representadas no mesmo plano, já que o número de incógnitas continua sendo dois (Figura 2.18). Para sistemas sobredeterminados, a solução não converge para um único ponto e irá oscilar dentro da vizinhança das diversas soluções

(interseção das retas de cada equação do sistema). Quando temos três incógnitas ou mais, não conseguimos reproduzir o que ocorre no plano (KAK; SLANEY, 2001).

Figura 2.18 – Ilustração do método de resolução do ART para um sistema 3x2.



Fonte: Adaptado de Kak e Slaney (2001).

Matematicamente, o vetor de vagarosidades na iteração i do processo ART é definido pela equação 2.17.

$$\vec{K}_i = \vec{K}_{i-1} + \Delta\vec{K} \tag{2.17}$$

A correção $\Delta\vec{K}$ é calculada à partir da equação 2.18 e o vetor $\vec{K}_i$ é alterado a cada projeção, que representa uma iteração do método.

$$\Delta\vec{K} = \frac{\left(\vec{K}_{i-1} \cdot \vec{S}_j - T_j\right)}{\vec{S}_j \cdot \vec{S}_j}\vec{S}_j \tag{2.18}$$

Fatores como ruídos de medição e ângulo formado entre cada uma das retas das equações do sistema influenciam na velocidade da convergência do método (KAK; SLANEY, 2001).

Por fim, tem-se a técnica de reconstrução iterativa simultânea (*Simultaneous Iterative Reconstruction Technique* - SIRT), também conhecido como Método de Cimmino, por ter sido inicialmente proposto por este (CIMMINO, 1938). O seu princípio de funcionamento é muito semelhante ao do ART: inicia-se com um palpite inicial para o vetor de vagarosidades $\vec{K}_i$, este vetor é projetado na primeira equação do sistema linear e calcula-se a correção $\Delta\vec{K}_j$ da medição, aplicando a equação 2.18.

No entanto, essa correção só é aplicada ao vetor de vagarosidades no final de uma iteração completa deste método, ou seja, após projetar o mesmo vetor $\vec{K}_i$ em todas as equações do sistema, calcular uma correção $\Delta\vec{K}_j$ para cada projeção e encontrar a média destes valores (HAACH, 2017). A equação 2.19 elucida o cálculo do novo vetor de vagarosidades ao final da iteração, em que m novamente representa o número de medições realizado (total de equações do sistema linear).

$$\vec{K}_i = \vec{K}_{i-1} + \frac{\sum \Delta\vec{K}_j}{m} \tag{2.19}$$

Como cada iteração do método é mais demorada, a sua velocidade de convergência é menor. Apesar disso, ele costuma gerar imagens melhores que o ART (KAK; SLANEY, 2001).

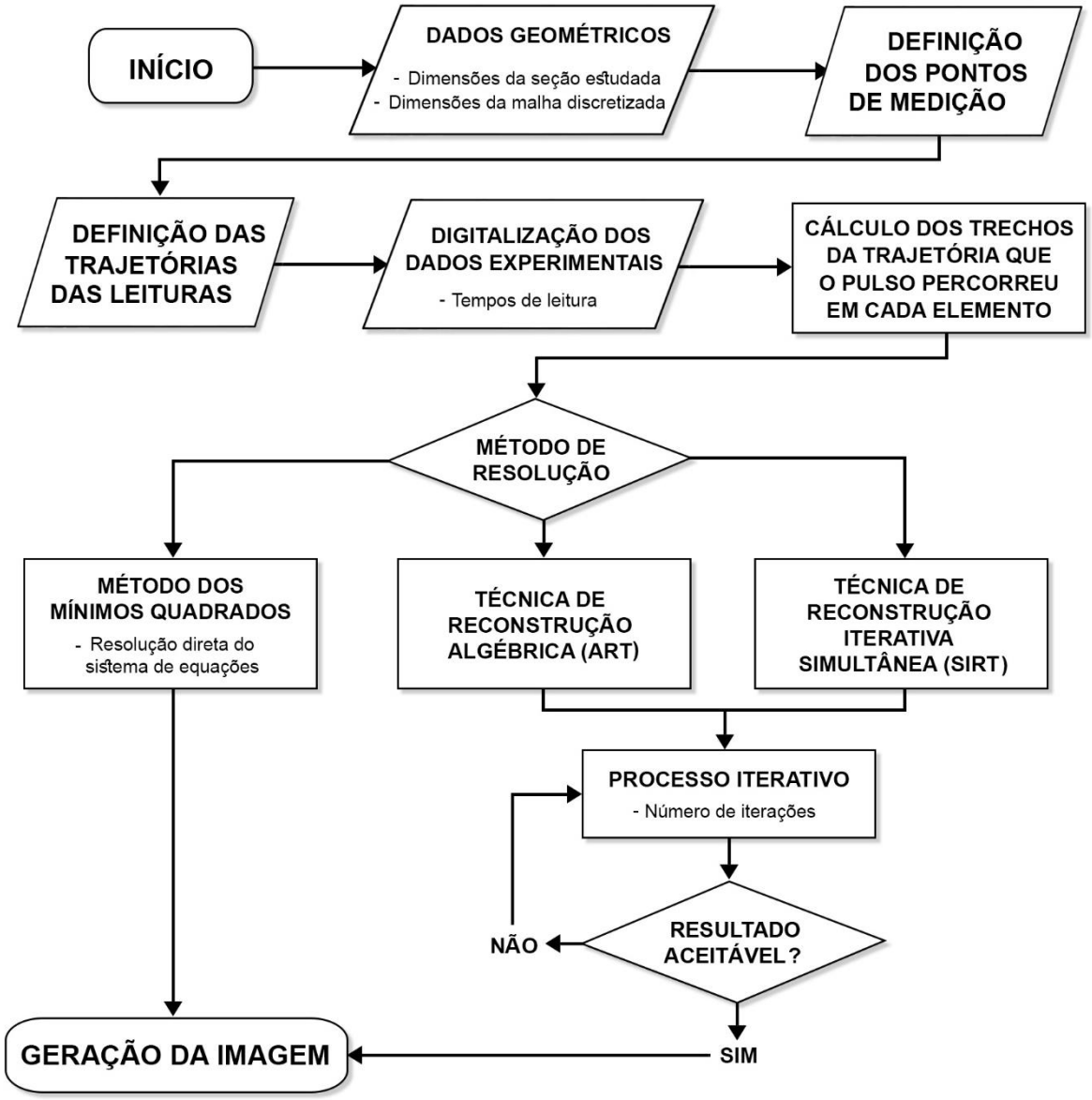
Após aplicar algum dos três métodos de solução (OLS, ART ou SIRT), encontram-se as vagarosidades de cada elemento, seguidas das vagarosidades de cada nó da malha. Como a velocidade e a vagarosidade são grandezas opostas, é possível obter a primeira também. Interpolando linearmente os valores de velocidades nodais, gera-se o mapa de velocidades do pulso ultrassônico na seção transversal estudada, em escala de cores contínua.

Além de explorar os pontos principais do *software* TUSom, montou-se o fluxograma da Figura 2.19, que ilustra resumidamente as etapas de seu funcionamento.

Buscando abandonar a hipótese de trajetórias retilíneas para o pulso ultrassônico, autores como Perlin e Pinto (2019) utilizaram-se do algoritmo

determinístico de Dijkstra para calcular as trajetórias ótimas entre transdutores na malha de nós discretizada.

Figura 2.19 - Fluxograma das etapas de funcionamento do TUSom.



Fonte: Adaptado de Ramirez (2015).

No presente trabalho, dois algoritmos determinísticos e dois algoritmos bioinspirados serão implementados computacionalmente para mapear as trajetórias de pulsos ultrassônicos. O primeiro grupo é representado pelos algoritmos de Dijkstra e A*, enquanto o segundo é representado pelos algoritmos genéticos (GAs) e de otimização por colônia de formigas (ACO). No entanto, antes de explorar o princípio de funcionamento de cada um deles, é necessário conhecer melhor alguns conceitos de ciência da computação.

2.5 CONCEITOS DE CIÊNCIA DA COMPUTAÇÃO

Inicialmente, o conceito de problema de busca será explorado. Em seguida, tratam-se dos grafos, uma representação comum para este tipo de problema. A análise de complexidade dos problemas de busca também será abordada. Essa análise se mostra crucial para a escolha de um agente de buscas, que irá resolver o problema. Este agente pode ter seu desempenho avaliado de diversas formas, também mencionadas na seção.

2.5.1 PROBLEMAS DE BUSCA

Um dos problemas mais elementares da ciência da computação são os problemas de busca, que tem como objetivo básico encontrar uma solução ótima para um problema, dado um espaço de soluções existente (LINDEN, 2006). Sua importância provém do fato de que grande parte dos problemas da vida real podem ser convertidos em problemas de busca.

Formalmente, os problemas de busca apresentam 5 componentes principais (RUSSEL; NORVIG, 2010):

a) Um espaço de estados finito que o problema pode assumir, onde cada estado é uma especificação de certos aspectos relevantes para o problema;
b) Um estado inicial, do qual o agente de buscas parte;
c) Ações que alteram os estados do problema, a serem desempenhadas pelo agente;
d) Um ou mais estados objetivo, determinados inicialmente;
e) Uma função de custo (ou objetiva ou de avaliação), que atribui um custo a cada ação. É opcional, sendo utilizada quando deseja-se comparar o custo de todos os estados objetivo (quando existe mais de um).

Ainda segundo os autores (2010, p. 67), “um caminho no espaço de estados é uma sequência de estados, conectados por uma sequência de ações”. Os agentes citados representam a estratégia adotada para o encontro de soluções (estados objetivo) em um problema de busca. Alguns exemplos de estratégias são desde buscas exaustivas (busca em largura e profundidade, por exemplo) a uma infinidade de algoritmos.

Um problema de busca muito famoso é do caixeiro viajante (*Traveling Salesman Problem* - TSP), onde tem-se um comerciante (caixeiro viajante) e um conjunto de cidades. Partindo de uma cidade inicial, deseja-se descobrir qual caminho passa por todas as cidades (uma única vez por cada e retornando à cidade inicial) e resulta no menor percurso possível. Este exemplo será explorado ao longo do trabalho devido a sua semelhança com o problema de obtenção das trajetórias ótimas do pulso ultrassônico.

Para o caso do TSP, o espaço de estados seriam todas as cidades passíveis de visita do caixeiro; o estado inicial representa a cidade de onde ele parte; as ações desempenhadas são viagens entre cada uma das cidades e os estados objetivo são caminhos em que todas as cidades foram visitadas apenas uma vez, retornando à cidade inicial. A função de custo representa a distância entre cada uma das cidades. Dentre os estados objetivo, o mais desejável é aquele que apresenta menor custo ao caixeiro viajante, ou seja, o caminho em que ele visita todas as cidades, cumprindo as condições impostas e percorrendo a menor distância possível (LINDEN, 2006).

No problema das trajetórias, o pulso ultrassônico assume o lugar do caixeiro viajante e os nós da malha discretizada em uma seção de concreto assumem o papel das cidades. No entanto, existem algumas diferenças entre os problemas: agora, não há retorno ao ponto inicial e nem a necessidade de passar por todos os nós (cidades), restrições que dificultam o TSP. A restrição de visitação única para cada nó permanece.

Nos problemas de busca em que se conhece todo o espaço de estados de antemão, caso do TSP, já que conhecemos todas as cidades para a qual o caixeiro pode viajar e as distâncias entre elas, este espaço costuma ser representado na forma de árvores de busca ou grafos.

2.5.2 GRAFOS

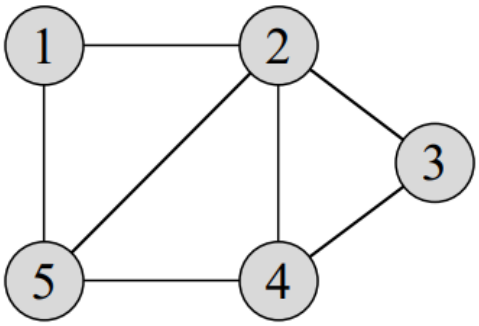
A teoria dos grafos é uma importante ramificação da matemática e tem sido estudada por centenas de anos. Propriedades foram descobertas, algoritmos foram desenvolvidos e muitos problemas continuam em estudo até hoje (SEDGEWICK; WAYNE, 2014). Em alguns trabalhos mais específicos da área de engenharia, essa área de estudo também é chamada de teoria das redes, do inglês *Network Theory*, em que um dos pioneiros foi Moser (1991), que estudou o traçado mais curto de raios

sísmicos que partem de uma origem e fazem parte de uma rede de pontos passíveis de visita, na área de geofísica, utilizando algoritmos de caminho mais curto, apresentados posteriormente. Esse problema é semelhante ao TSP e ao problema das trajetórias, já explorados neste trabalho.

Um grafo é uma estrutura de armazenamento de dados que guardam relações entre si, possuindo vértices (ou nós), que são numerados/nomeados. As ligações entre cada par de vértices são chamadas arestas e estabelecem relações entre eles, podendo apresentar pesos positivos ou negativos (ROMAN, 2017a). Geralmente utiliza-se a notação G (V, A) para representar um grafo G que possui um número V de vértices e A de arestas (CORMEN et al., 2009).

Na Figura 2.20, um grafo G(5,7) sem pesos foi reproduzido graficamente. Quando isso ocorre, os vértices costumam ser representados por círculos e numerados, enquanto as arestas costumam ser representadas por linhas, que ligam estes vértices.

Figura 2.20 – Grafo sem pesos com 5 vértices e 7 arestas.



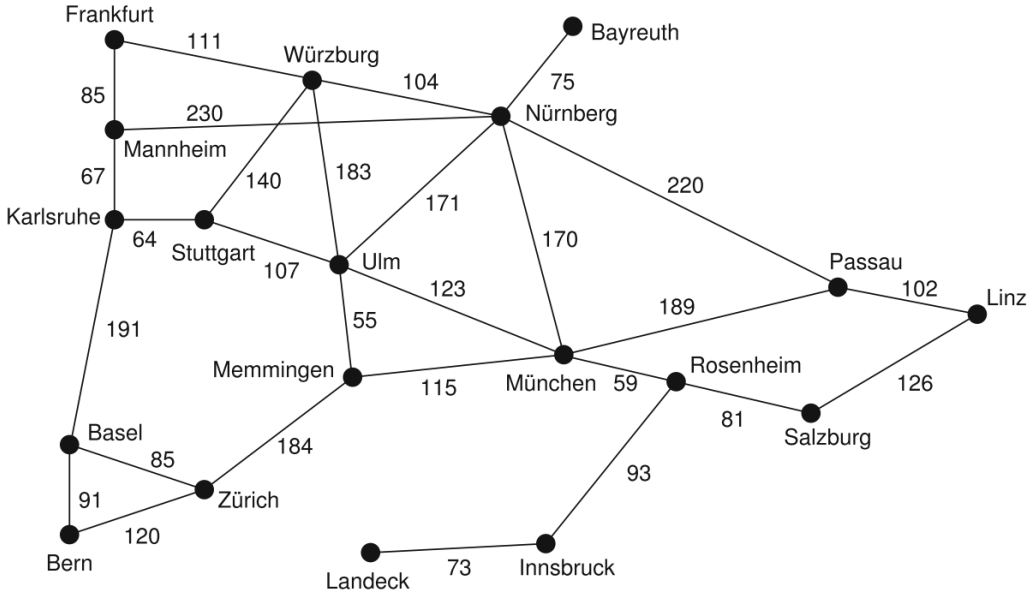
Fonte: Cormen et al. (2009).

Adotando a nomenclatura formal da seção 2.5.1 para os grafos, seus vértices representam os estados de um problema de busca, suas arestas representam possíveis ações e os pesos dessas arestas representam uma função de custo associada à cada ação. O conjunto de todos os vértices do grafo forma o espaço de estados possíveis.

Quando se representa o TSP através de um grafo, temos resultado semelhante a um mapa: os vértices indicam as cidades, enquanto as arestas indicam os caminhos entre cidades, ponderados por seus pesos, nesse caso positivos e que representam a distância em quilômetros entre cada cidade (SEDGEWICK; WAYNE, 2014). Esse problema é de grande interesse para empresas de logística, por exemplo, e é

representado na Figura 2.21 para a região do sul da Alemanha e partes da Suíça e Áustria.

Figura 2.21 – Representação gráfica de um grafo com as cidades e distâncias do sul da Alemanha e partes da Suíça e Áustria.



Fonte: Ertel (2017).

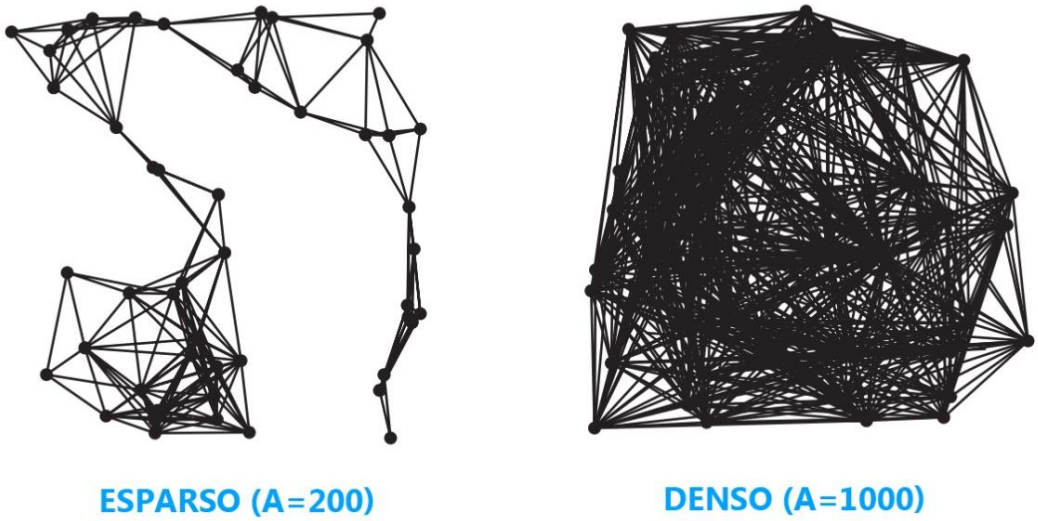
A nomenclatura dentro da teoria dos grafos é muito vasta e apenas os principais termos serão definidos. Os grafos são chamados dirigidos quando apresentam um sentido único de deslocamento (apenas ida) ou não dirigidos quando isto não ocorre (ida e volta) (SEDGEWICK; WAYNE, 2014). A Figura 2.21 é um exemplo de grafo não dirigido, pois podemos ir de Frankfurt a Mannheim (cidade adjacente logo abaixo) ou vice-versa e a distância percorrida será a mesma, o que também ocorre para as outras cidades.

Um grafo ainda é chamado esparso se o número de arestas (A) é muito menor que o quadrado do número de vértices (V^2). Quando o número de arestas se aproxima do quadrado de vértices, ele é chamado denso (CORMEN et al., 2009). A Figura 2.22 apresenta um exemplo de dois grafos com o mesmo número de vértices (50), porém diferentes em seu número de arestas: um deles é esparso, com apenas 200, enquanto o outro é denso, possuindo 1000 arestas.

Computacionalmente, um grafo costuma ser implementado de duas formas: através de listas ou matrizes de adjacência. A primeira consiste na construção de um vetor para cada vértice do grafo, que informa com quais vértices este se conecta

(chamados vértices adjacentes). Esta abordagem ocupa uma memória da ordem de V+A, geralmente menor. Por esse motivo, é mais utilizada, principalmente no caso de grafos esparsos.

Figura 2.22 – Comparação entre um grafo esparso e um denso com a mesma quantidade de vértices (50).



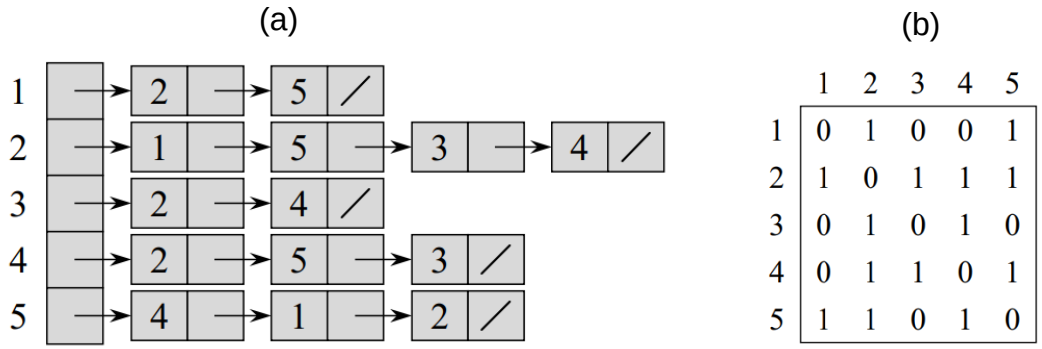
Fonte: Sedgewick e Wayne (2014).

A segunda abordagem consiste na construção de uma matriz de dimensões VxV, requerendo portanto esta memória. Dessa forma, cada linha da matriz representa um vértice e cada coluna representa outro vértice ao qual este pode se conectar. Quando essa conexão existe, o termo dessa coluna indica o peso da ligação (função de custo). Caso contrário, o termo possui valor nulo ou inválido. Apesar de consumir mais memória, é a melhor representação para grafos densos, dada a facilidade de indexação, que fornece rapidamente a informação de conexão ou não entre dois vértices e o custo da conexão entre eles (CORMEN et al., 2009).

A Figura 2.23 apresenta as duas formas de implementação para o grafo da Figura 2.20. Como o grafo em questão não possui pesos, se um vértice i se conecta a um vértice j, tem-se o valor 1 no termo i,j e j,i da representação matricial (como o grafo não é dirigido, tanto a ida quanto a volta são possíveis e possuem o mesmo custo). Caso a conexão entre vértices não exista, o valor do termo é nulo.

Cada problema de busca possui uma estrutura própria, alterando os seus componentes principais apresentados na seção 2.5.1 e a sua forma de representação. Para escolher a melhor implementação e um agente de buscas adequado para a sua resolução, é interessante que se faça uma análise de complexidade do problema.

Figura 2.23 – Formas de implementação computacional mais comuns para o grafo mostrado anteriormente – G(5,7).



a) Implementação por listas de adjacência; b) Implementação por matriz de adjacência.
Fonte: Cormen et al. (2009).

2.5.3 ANÁLISE DE COMPLEXIDADE

O campo de análise de complexidade separa os problemas de busca em duas categorias principais: os que podem ser resolvidos em tempo polinomial, chamados problemas P (*Polynomial Problems*) ou fáceis, e os que não podem ser resolvidos em tempo polinomial, mas cuja solução pode ser verificada em tempo polinomial, chamados problemas NP (*Nondeterministic Polynomial Problems*) ou difíceis (RUSSEL; NORVIG, 2010). Essa classificação vale para qualquer que seja o agente de busca utilizado na solução do problema.

Os problemas NP constituem uma classe de problemas de busca muito presente em diversos ramos da ciência. O TSP faz parte deste grupo, pertencendo ainda ao subgrupo dos problemas chamados NP-Completos, onde a palavra “completo” possui sentido de “mais extremo”, denotando os problemas mais complexos dentro da classe NP.

Para entender o significado prático do crescimento de dados de cada tipo de problema de busca, compara-se essa proporção para exemplos de problemas P e NP na Tabela 2.2, onde n representa o tamanho da entrada de dados do problema.

Voltando ao caso do TSP, n representa o número de cidades a serem visitadas. Supondo que o algoritmo se inicia em qualquer uma das n cidades, restam n-1 opções para visita. Partindo para a próxima cidade, tem-se n-2 restantes, e assim em diante, até que se visite a última. Desta forma, conclui-se que a quantidade de opções de estados objetivo que o TSP apresenta são (n-1) x (n-2) x ..., ou seja, crescem em proporção fatorial (n-1)!. Isso significa que, para um arranjo de 101 cidades, tem-se

aproximadamente 10^{158} soluções possíveis. Tendo em consideração que uma máquina moderna é capaz de realizar cerca de 10^9 operações por segundo, a busca exaustiva da solução ótima dentre todas possíveis levaria cerca de 10^{141} anos (LINDEN, 2006).

Tabela 2.2 – Crescimento de dados em problemas do tipo P e NP.

Problemas P			**Problemas NP**	
n	n^2	n^3	2^n	$n!$
10	10^2	10^3	$\approx 10^3$	$\approx 10^6$
100	10^4	10^6	$\approx 10^{30}$	$\approx 10^{158}$
1000	10^6	10^9	$\approx 10^{300}$	$\gg 10^{1500}$
10000	10^8	10^{12}	$> 10^{3000}$	$\gg 10^{10000}$

Fonte: Linden, 2006.

Observando a Tabela 2.2 e o exemplo do TSP, pode-se concluir que os problemas NP apresentam um ritmo de crescimento de dados muito maior que o dos problemas P. Desta forma, o tipo de problema deve ser levado em consideração no momento da escolha do agente de buscas, visto que o seu tempo de solução cresce de forma proporcional aos dados, podendo se tornar o que chamamos de problema intratável quando o tempo de solução (um dos parâmetros de desempenho dos agentes de busca) é inaceitável ao usuário (LINDEN, 2006).

2.5.4 MEDIDAS DE DESEMPENHO DE AGENTES DE BUSCA

Juntamente com outros três parâmetros, o tempo forma o conjunto de medidas de desempenho de um agente de buscas (RUSSEL; NORVIG, 2010):

a) Completude: diz respeito à capacidade do agente de busca de sempre encontrar algum estado solução para o problema, caso esse exista;
b) Otimalidade: se a solução encontrada pelo agente é sempre a ótima;
c) Tempo: mede quanto tempo o agente demora para encontrar uma solução;
d) Espaço: mede quanta memória é necessária para que o agente realize a busca.

No que diz respeito aos agentes de busca que serão utilizados no presente trabalho, todos estes fatores são de suma importância, com exceção do espaço,

considerado secundário em um primeiro momento, desde que não exceda a quantia de memória disponível no equipamento que irá executar o *software* TUSom. Os agentes de busca são representados por algoritmos e cada um destes algoritmos e seus princípios de funcionamento serão explorados com maiores detalhes nas próximas seções.

2.6 ALGORITMOS DETERMINÍSTICOS

No presente trabalho, os agentes de busca tomam a forma de algoritmos determinísticos ou bioinspirados. Num problema de busca, diz-se que um algoritmo é determinístico se determinada ação tomada em determinado estado leva sempre à mesma resposta (outro estado específico) (ERTEL, 2017). Dessa forma, não importa quantas vezes este algoritmo seja executado: para determinada entrada de um problema, ele sempre irá fornecer a mesma saída. Estão representados neste trabalho pelos algoritmos de Dijkstra e A*.

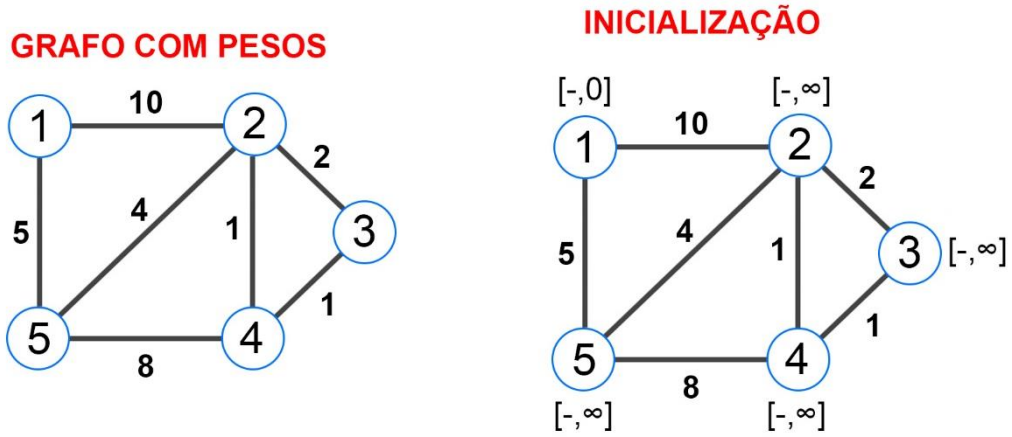
2.6.1 ALGORITMO DE DIJKSTRA

O algoritmo foi concebido por Dijkstra (1959) e é aplicado para encontrar o caminho de custo mínimo em grafos dirigidos ou não quando os pesos das arestas são positivos, partindo-se de um vértice inicial especificado. Essa classe de problemas de busca é conhecido como caminho mínimo de fonte única (*single-source shortest paths*) (SEDGEWICK; WAYNE, 2014). No caso do TSP, como tratamos de distâncias entre cidades, os pesos sempre assumem valores positivos, se enquadrando perfeitamente na classe de problemas resolvida pelo algoritmo.

Na Figura 2.24, o algoritmo pretende encontrar caminhos mais curtos (menor custo possível) no grafo da Figura 2.20, ao qual foram atribuídos pesos. Inicialmente, todos os vértices do grafo fazem parte do grupo de “não visitados”. Primeiramente, seleciona-se o vértice desejado para ser a origem (O), o vértice 1 no caso da figura, e outro vértice para ser o destino (D), o vértice 4 na figura. Em seguida, dá-se um palpite inicial para a distância entre o vértice 1 e os demais (processo de inicialização), em que a distância entre 1 e ele mesmo é considerada nula e entre 1 e todos outros vértices é considerada “infinita” (computacionalmente, é representado como o maior valor real possível).

Além disso, o precedente que gera a menor estimativa de distância também é armazenado, a fim de traçar o melhor caminho entre os vértices 1 e 4 ao final da aplicação do algoritmo. Na inicialização do algoritmo, os pontos precedentes são considerados como um valor nulo/vazio. O par ordenado [-,∞] indica o vértice precedente que gerou a menor distância (vazio, pois nenhum vértice foi visitado) e a menor estimativa de distância do vértice em questão à origem, respectivamente.

Figura 2.24 - Exemplo de aplicação do algoritmo de Dijkstra a um grafo - inicialização.

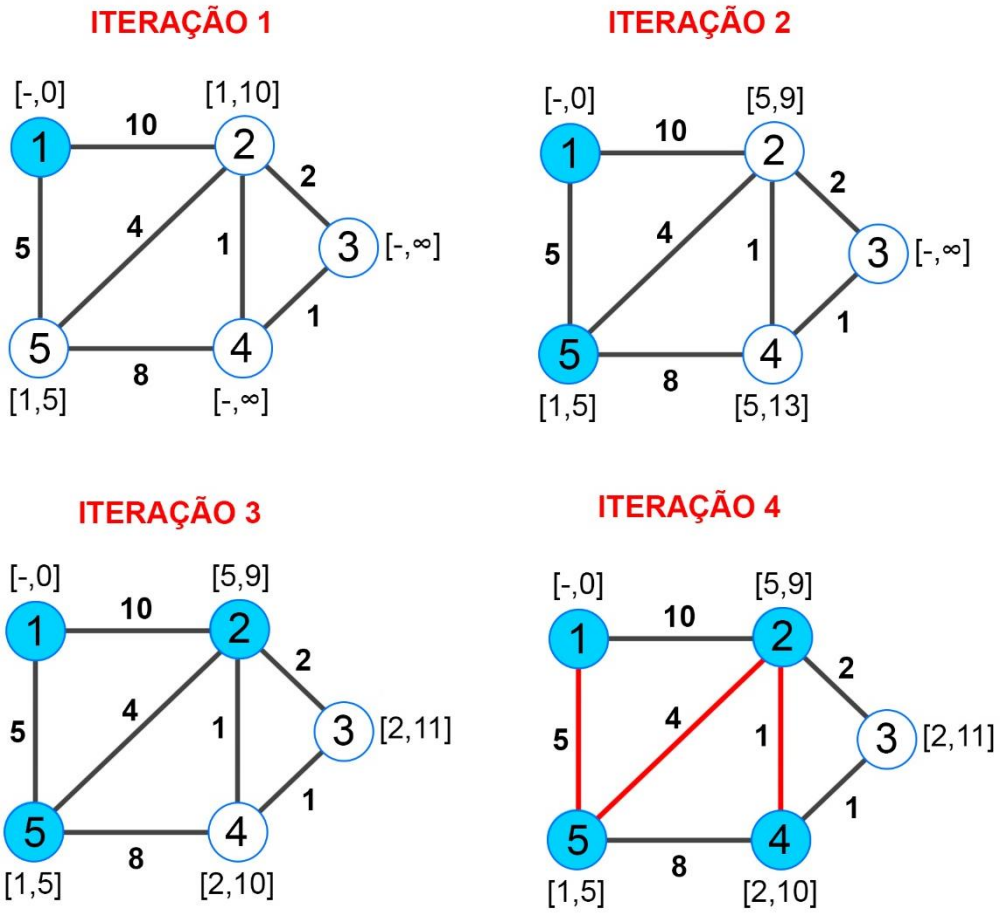


Fonte: Autor.

Em seguida, inicia-se o processo iterativo do algoritmo (ilustrado na Figura 2.25): seleciona-se o vértice à menor distância da origem. Como a distância entre o vértice 1 e ele mesmo é nula, este é selecionado. Marca-se este como “visitado” (cor azul na figura) e todos os vértices adjacentes (vizinhos) a ele são analisados. No caso da na Figura 2.25, os vértices adjacentes a 1 são 2 e 5.

É avaliado se o custo da origem até 1, somado ao custo entre 1 e os adjacentes (conhecido, pois representa o peso da aresta entre 1 e cada vértice) é menor que a estimativa atual. Nesse caso, se atualiza a estimativa de distância entre os dois vértices. Esse processo é chamado de relaxamento da aresta. Caso o novo custo seja maior que o atual, nada é feito. No exemplo, como o custo entre 1 e 5 é de 5, menor que “infinito”, a estimativa inicial, adota-se esse novo valor no lugar do anterior. O mesmo ocorre para o vértice 2: como o custo entre 1 e 2 é 10 e este é menor que “infinito”, a nova estimativa de custo entre 1 e 2 é 10.

Figura 2.25 - Exemplo de aplicação do algoritmo de Dijkstra a um grafo - iterações.



Fonte: Autor.

Posteriormente, seleciona-se o próximo vértice à menor distância da origem e este é marcado como visitado (vértice 5 no exemplo). Para este novo vértice, a distância ótima à origem já foi encontrada, além do precedente que fornece esse caminho (custo 5 até a origem e precedente 1). Agora, estimam-se as distâncias entre ele e seus vértices adjacentes (passando pelo caminho ótimo), relaxam-se as arestas caso custos menores sejam encontrados e o processo se repete até que atinja-se o destino (vértice 4).

É importante notar que não é necessário visitar todos os vértices do grafo, a não ser que se deseje conhecer a menor distância entre todos e a origem. No momento em que visita-se o destino, tem-se a menor estimativa entre este e a origem (destacado em vermelho na iteração 4 da Figura 2.25).

Após a aplicação do algoritmo, tem-se também a trajetória que gerou a menor distância entre a origem e o destino. Isso ocorre pelo armazenamento dos pontos

precedentes a todos os vértices (p[V]) que geraram a menor estimativa de distância entre este vértice e a origem (ROMAN, 2017b).

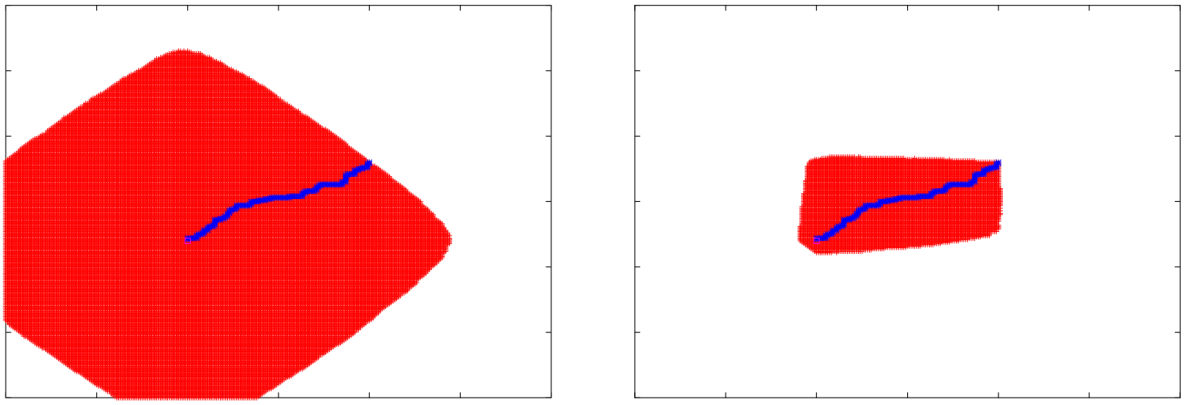
2.6.2 ALGORITMO A*

O algoritmo de Dijkstra pertence à categoria de algoritmos de busca não informada, o que significa que ele não possui informações sobre cada estado que o problema assume (cada vértice do grafo visitado ao longo da aplicação do algoritmo). Ele é capaz apenas de comparar a qualidade entre estados objetivo quando estes são atingidos, o que faz com que ele garanta a resposta ótima sempre (possui otimalidade).

Enquanto isso, o algoritmo A* pertence à categoria de algoritmos de busca informada, apresentando a capacidade de estimar, em todos os estados que não são objetivos, o custo até o objetivo, explorando apenas os estados que se mostram mais promissores no sentido de alcançá-lo. Essa diferença faz com que ele visite menos vértices que o algoritmo de Dijkstra, apresentando potencial para um desempenho superior (RUSSEL; NORVIG, 2010).

A Figura 2.26 apresenta uma comparação entre a aplicação dos algoritmos de Dijkstra e A* para uma mesma entrada de dados, onde existe um ponto de origem e um ponto de destino. Em vermelho, é mostrada uma nuvem que representa a área vasculhada por cada um dos algoritmos até que a solução ótima (em azul marinho) fosse encontrada por cada um.

Figura 2.26 – Comparação do número de vértices visitados pelos algoritmos de Dijkstra e A* entre um ponto de origem e um ponto de destino, para a mesma entrada.



Fonte: Adaptado de Goldberg e Harrelson (2004).

Para avaliar quais estados do problema de busca apresentam maior potencial de chegar ao objetivo com menor custo, utiliza-se uma função de avaliação $f(V)$, composta por duas parcelas principais (MICHALEWICZ; FOGEL, 2004), vide equação 2.20: uma parcela $c(V)$, que avalia a qualidade (custo) do caminho até o estado atual, e uma parcela $h(V)$, chamada heurística, que avalia o potencial dos estados restantes atingirem o objetivo. A heurística incorpora conhecimentos adicionais sobre o problema, diferenciando esse tipo de algoritmo dos de busca não informada (RUSSEL; NORVIG, 2010).

$$f(V) = c(V) + h(V) \tag{2.20}$$

A função de avaliação é calculada a cada iteração do algoritmo. Para o caso do TSP, quando representado por um grafo, $c(V)$ apresenta uma interpretação direta, indicando a distância total (custo) do caminho percorrido, da origem até o vértice atual. Enquanto isso, $h(V)$ é calculada através de alguma estratégia adotada pelo usuário, como a distância em linha reta entre o vértice atual e o vértice objetivo. Mesmo que esse caminho não seja possível nos moldes do grafo em questão, ele serve como uma estimativa de qualidade das soluções atuais. Como desejamos escolher o caminho de menor custo (o TSP é um problema de minimização), o caminho que possui a menor função de avaliação (menor distância à origem) é o mais desejável e será escolhido para a próxima visita (assim como era feito em Dijkstra).

Uma propriedade importante do algoritmo A* é que a heurística h vale 0 para o vértice objetivo, visto que representa a estimativa de distância entre dois pontos localizados no mesmo lugar (ERTEL, 2017).

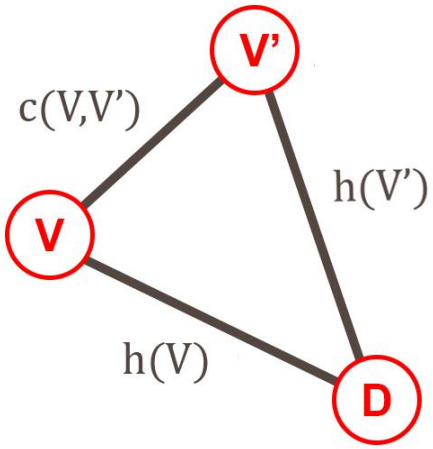
A lógica utilizada pelo algoritmo A* é a mesma do algoritmo de Dijkstra, ou seja, escolhem-se sempre os vértices que geram caminhos com a menor função de avaliação como os próximos a serem visitados. A diferença consiste no fato de que, no algoritmo de Dijkstra, não há a consideração da heurística, o que corresponderia ao caso em que $f(V) = c(V)$.

No que diz respeito às medidas de desempenho descritas na seção 2.5.4, tanto o algoritmo de Dijkstra quanto o algoritmo A* são completos e ótimos, ou seja, sempre encontram a melhor solução no espaço de buscas, quando ela existe. No entanto, para que isso ocorra no algoritmo A*, deve-se cumprir a condição de uma heurística consistente.

Considerando-se um vértice V, o seu sucessor V’ e a ação A que leva V a V’, uma heurística é dita consistente quando o custo estimado do vértice V ao objetivo (destino D) é sempre menor que o custo estimado do vértice V’ ao objetivo somado ao custo da ação A, conforme a equação 2.21 e ilustrado na Figura 2.27. É possível observar que a consistência de uma heurística é uma expressão da desigualdade triangular. Esta afirma que um dos lados de um triângulo nunca pode ser maior que a soma dos seus dois lados restantes (RUSSEL; NORVIG, 2010).

$$h(V) \leq c(V, V') + h(V') \tag{2.21}$$

Figura 2.27 – Condição necessária a uma heurística consistente (desigualdade triangular).



Fonte: Autor.

Quando bem implementado, o algoritmo A* promove uma diminuição considerável do espaço de busca, compensando o custo computacional do cálculo da heurística. No entanto, é necessário um bom gerenciamento da memória, visto que o algoritmo armazena todos os vértices explorados (RUSSEL; NORVIG, 2010).

Ainda que o espaço de busca diminua consideravelmente no algoritmo A*, o tempo de processamento dos algoritmos determinísticos cresce de forma exponencial com relação ao tamanho do problema, podendo não apresentar desempenho satisfatório. Tendo isso em vista, os algoritmos bioinspirados surgem como novas alternativas à solução de problemas de busca, fornecendo soluções aproximadas.

2.7 ALGORITMOS BIOINSPIRADOS

Os algoritmos bioinspirados são ferramentas utilizadas para solução de problemas nos mais variados ramos da ciência, tendo origem na década de 1970 (HAUPT; HAUPT, 2004). Sua criação parte da observação de fenômenos da natureza otimizados e bem sucedidos e tem beneficiado sobretudo o campo de inteligência computacional, criando algoritmos capazes de simular comportamentos inteligentes, antes vistos como exclusivamente humanos. Aliados ao crescimento do poder computacional das máquinas modernas, os algoritmos bioinspirados conseguem encontrar soluções para problemas extremamente complexos (LINDEN, 2006).

No presente trabalho, serão abordadas duas alternativas de algoritmos bioinspirados, que serão aplicadas na solução do problema de determinação das trajetórias do pulso ultrassônico, já enquadrado na abrangente classe dos problemas de busca. Estas alternativas são os algoritmos genéticos (simulação da evolução natural) e a otimização por colônia de formigas (simulação do processo de busca de alimentos por formigas) (LINDEN, 2006).

Estes algoritmos aplicam operadores probabilísticos a uma população de indivíduos, rumando estatisticamente para os pontos de mínimo/máximo de uma função predeterminada. Neles, não há necessidade de derivadas, o que diminui drasticamente o problema de convergência a mínimos locais (HAUPT; HAUPT, 2004).

No geral, estes algoritmos são muito simples e não apresentam grandes dificuldades de implementação computacional. No entanto, precisam ser muito bem estudados, pois apresentam diversos parâmetros e variações de estrutura que devem ser ajustados às necessidades do problema que se deseja resolver.

2.7.1 ALGORITMOS GENÉTICOS (GAs)

Os GAs foram desenvolvidos por John Holland em 1975 e consistem em uma técnica de busca e otimização baseada nos princípios da seleção natural. Dessa forma, para entender completamente o seu funcionamento, é necessário compreender alguns conceitos dessa teoria (HAUPT; HAUPT, 2004). Resumidamente, pode-se dizer que a seleção natural surgiu à partir do conjunto de estudos desenvolvidos por Darwin, Mendel e Lamarck no século XIX. A teoria prega que os indivíduos melhor adaptados ao meio são aqueles que sobrevivem.

Antes de prosseguir, é importante também definir alguns conceitos de genética:

a) Genes: constituem a unidade básica da hereditariedade, carregados nos seres vivos em pares de cromossomos, que compõe a estrutura genética do organismo (Ácido Desoxirribonucleico - DNA);
b) Cromossomos: são pacotes de genes. Cada espécie de organismo possui um número determinado de cromossomos que estão presentes no núcleo de todas as células que o compõe (LIMA, 2011). O ser humano, por exemplo, possui 23 pares de cromossomos;
c) População: é um conjunto de indivíduos que convivem num mesmo ambiente em determinado espaço de tempo;
d) Reprodução sexual: tipo de reprodução comum entre os seres vivos mais complexos, onde os gametas masculino e feminino, cada um com metade da carga de cromossomos do indivíduo, se misturam, gerando um novo ser com um misto de características dos seus progenitores;
e) Mutação: alteração definitiva do material genético de um ser que ocorre de forma espontânea ou devido a estímulos do ambiente. Este ser passa a manifestar uma nova característica, não proveniente dos pais, que pode ser transmitida às próximas gerações (LIMA, 2011);
f) Aptidão biológica: medida do quão adaptado um ser é ao ambiente em que vive, podendo ser função de uma ou mais características desejáveis (HAUPT; HAUPT, 2004).

Além da genética, no que diz respeito à evolução, os indivíduos mais aptos conseguem se reproduzir com maior frequência, gerando uma população cada vez mais preparada ao ambiente (HAUPT; HAUPT, 2004). Como os indivíduos menos aptos se reproduzem menos, são cada vez em menor número e pouco a pouco vão sendo substituídos pelos mais aptos.

No algoritmo genético, as características de um ambiente natural são simuladas. Resumidamente, o algoritmo pode ser descrito da seguinte maneira: gera-se aleatoriamente uma população composta por n indivíduos de uma mesma espécie, cada qual com suas características, determinadas pelos genes. Os genes são representados matematicamente por valores em um domínio predeterminado. É

importante ressaltar que as palavras “indivíduo” e “cromossomo” são empregadas como sinônimos na linguagem dos GAs, indicando um conjunto de genes.

Em seguida, atribui-se um valor de aptidão biológica (*fitness*) a cada indivíduo, de acordo com uma função que considera características desejáveis para o problema (LIMA, 2011). Assim, os indivíduos são candidatos a solução de um problema e representam, por exemplo, entradas x de uma função f(x). Tem-se então a primeira geração de indivíduos.

À partir disso, aplica-se um critério de seleção dos melhores indivíduos, visando escolher os que irão se reproduzir e consequentemente propagar seus genes à próxima geração. O mais comum é o critério da roleta, onde a seleção é feita por sorteio e cada indivíduo possui uma probabilidade de seleção proporcional a sua aptidão. Essa probabilidade é descrita pela equação 2.22. A variável $P(b_j)$ indica a probabilidade do indivíduo b_j ser selecionado, n indica o tamanho da população e $f(b_j)$ indica o valor da função de aptidão para o indivíduo b_j (RIBEIRO, 2019).

$$P(b_j) = \frac{f(b_j)}{\sum_{i=1}^{n} f(b_i)} \tag{2.22}$$

Depois da seleção, sorteia-se um número aleatório, e se este for menor que uma probabilidade de cruzamento (PC) definida, é feito o cruzamento (*crossover*) entre dois indivíduos, selecionados de forma aleatória também ou aplicando algum critério de seleção. O cruzamento é o operador responsável pela combinação dos genes de dois indivíduos (pais), de forma a gerar novos indivíduos (filhos) com as características herdadas dos mesmos, simulando a reprodução sexual. Existem três tipos principais de cruzamento (RIBEIRO, 2019):

a) Cruzamento “um-ponto”: à partir de dado ponto, as características dos pais são trocadas;
b) Cruzamento “multiponto”: a troca de genes utiliza vários pontos;
c) Cruzamento “uniforme”: determinam-se quais genes de cada um dos pais será herdado por cada um dos filhos.

Para que a população não se torne homogênea após algumas gerações, existe uma probabilidade de mutação (PM) associada ao algoritmo. Após a seleção e cruzamento, sorteia-se um número aleatório e, se este for menor que PM, a mutação ocorre, alterando genes aleatórios de determinado indivíduo. O parâmetro PM deve

ser sempre menor que PC e ambas as probabilidades são ajustadas experimentalmente de acordo com o problema que se deseja resolver (RIBEIRO, 2019). Assim como o cruzamento, existem muitos modelos de mutação.

Quando não se deseja perder determinados indivíduos mais aptos, é possível transmiti-los diretamente à próxima geração sem a necessidade de passar pelo processo de seleção e cruzamento. Esse processo é chamado de elitismo. Quando se desejam aplicar restrições à função de aptidão, utiliza-se a penalização. Assim, quando determinado indivíduo possui uma característica indesejável, ele perde parcela de sua aptidão, diminuindo suas chances de se reproduzir (LIMA, 2011).

Com o passar dos anos, concluiu-se que o custo computacional da aplicação do elitismo compensava a melhora substancial dos resultados de cada geração, garantindo que esta seja no mínimo tão boa quanto a anterior (LINDEN, 2006).

Após todas as suas etapas (seleção, cruzamento, mutação e elitismo – sendo o último opcional), o processo se repete até que determinado critério de parada seja atingido, podendo ser um valor para o erro, um determinado número de gerações ou até um número de gerações sem que tenham-se alterações no melhor indivíduo. O fluxograma da Figura 2.28 demonstra o esquema simplificado do funcionamento de um algoritmo genético.

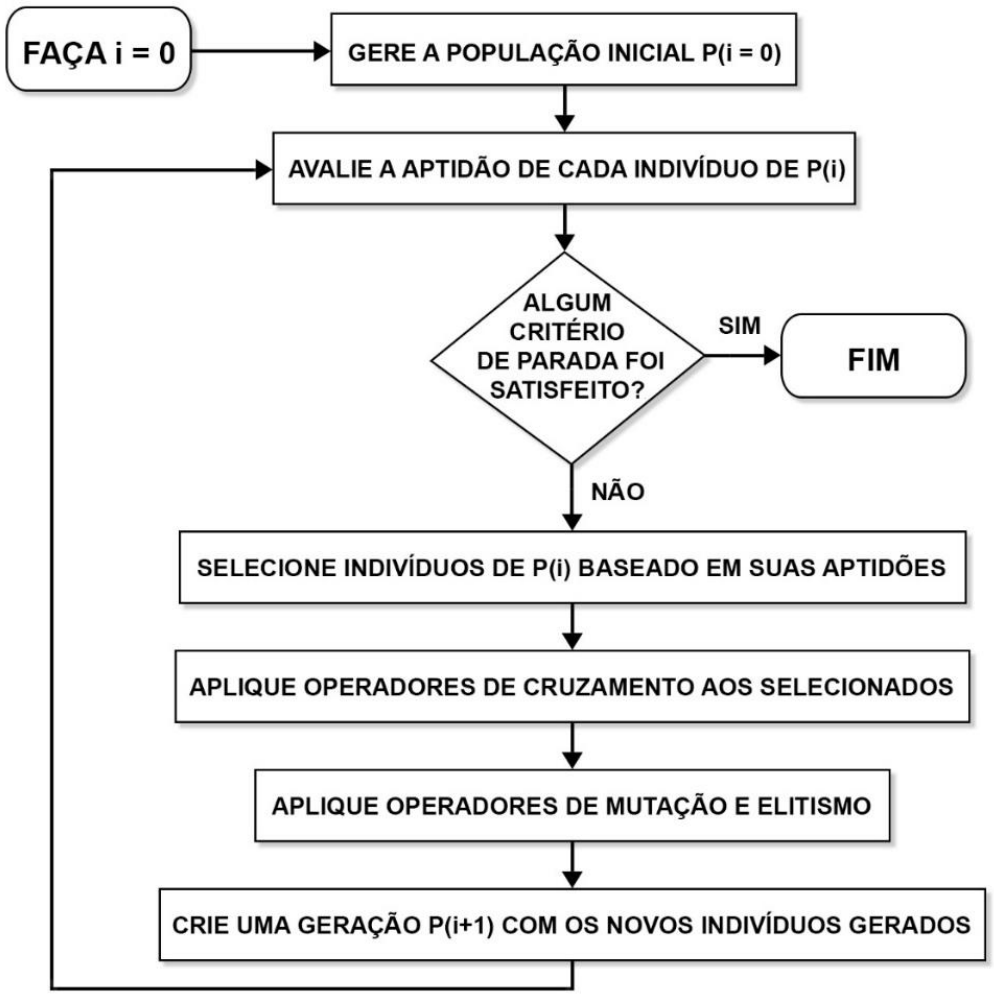
No que diz respeito à implementação computacional, o primeiro algoritmo genético, proposto por Holland (1992), apresentava representação binária e três operadores genéticos essenciais: seleção, cruzamento e mutação. No entanto, a representação dos indivíduos deve ser a mais natural possível, adequando a ferramenta ao problema e não o contrário. Representações inteiras, reais ou até letras são possíveis no GA (LINDEN, 2006).

A eficiência dos algoritmos genéticos em encontrar máximos/mínimos globais para funções reside no fato do cruzamento efetuar uma busca local de soluções (*explotation*), enquanto a mutação efetua buscas globais menos frequentes (*exploration*). No entanto, como o método se baseia em processos estocásticos, cada execução produz uma resposta diferente e não existe a garantia de que a melhor resposta (máximo/mínimo global) sempre será encontrada (LINDEN, 2006).

Assim como os algoritmos apresentados anteriormente, os algoritmos genéticos também podem ser aplicados ao TSP. Voltando-se a aplicações em engenharia, algoritmos genéticos são frequentemente utilizados na determinação da navegação de robôs. Na engenharia civil, especificamente, Lima (2011) utilizou GAs

para a otimização topológica e paramétrica de vigas de concreto armado. Ribeiro (2019) utilizou GAs e redes neurais artificiais para a calibração de modelos de elementos finitos, visando a detecção de danos e anisotropia em estruturas de concreto.

Figura 2.28 – Fluxograma das etapas de funcionamento do GA.



Fonte: Adaptado de Lima (2011).

2.7.2 OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS

O método foi formulado e apresentado pela primeira vez por Dorigo (1992), batizado inicialmente de *Ant System* (AS), com o intuito de resolver problemas de otimização discreta. Juntamente com os GAs e outras técnicas não apresentadas neste trabalho, como a otimização por enxame de partículas (*Particle Swarm Optimization* – PSO), constituem as chamadas meta-heurísticas, que são “estratégias de alto nível que guiam uma heurística durante o processo de busca de um dado

problema” (CALVO, 2012, p. 30), evitando a convergência prematura a mínimos locais.

As meta-heurísticas se enquadram em duas categorias de métodos de busca: busca baseada em instância (*instance-based search*) ou busca baseada em modelos (*model-based search*). Em ambos os casos, os estados candidatos a solução são gerados através de um modelo probabilístico. No entanto, para o primeiro, apenas o estado atual é considerado na geração de novos estados. Como exemplo, tem-se os GAs sem elitismo.

Na segunda categoria, da qual a otimização por colônia de formigas (do inglês *Ant Colony Optimization* – ACO) e os GAs com elitismo fazem parte, o modelo é atualizado à partir de informações de todas as soluções previamente descobertas, de forma que a busca se concentre em regiões contendo apenas soluções de boa qualidade. Isso é feito através da alocação de uma parcela auxiliar de memória, além da utilizada pelo próprio método, para armazenar informações como a avaliação média das soluções ou as melhores soluções encontradas até então (ZLOCHIN et al., 2004).

Antes de definir os algoritmos ACO, é necessário conhecer os tipos de problema que ele resolve, chamados problemas de otimização combinatorial. Eles são tipos muito comuns de problemas de otimização que consistem em encontrar valores para variáveis discretas (quantitativas e não contínuas) que tornem ótimo o valor de uma função objetivo predeterminada. Esse valor ótimo diz respeito à uma minimização ou maximização. Os problemas de caminho mínimo de fonte única (*single-source shortest paths*), já citados na seção 2.6.1, pertencem à esse grupo (DORIGO; STÜTZLE, 2004).

Definindo-os formalmente, tem-se que uma instância de um problema de otimização Π (condição em que o problema possui valores especificados para cada parâmetro) é a tripla (S, f, Ω), em que S é um conjunto de candidatos a solução do problema (espaço de estados), f é uma função objetivo que designa um valor $f(s)$ para cada candidato $s \in S$ (função de custo) e Ω é um conjunto de restrições a ser respeitado no problema (DORIGO; STÜTZLE, 2004).

As soluções pertencentes ao conjunto $\tilde{S} \subseteq S$ que satisfazem as restrições impostas por Ω são chamadas de soluções viáveis. O objetivo principal neste tipo de problema é encontrar uma solução viável s^* que represente um ótimo global da função

objetivo f. Quando falamos de problemas de minimização, queremos encontrar $s^* \in \tilde{S}$ com custo mínimo, ou seja, uma solução tal que $f(s^*) \leq f(s)$ para todo $s \in \tilde{S}$. Quando tratamos de problemas de maximização, buscamos o custo máximo (DORIGO; STÜTZLE, 2004).

Por envolver um grande número de combinações, a instância de um problema de otimização combinatorial não costuma ser representado enumerando-se todos os candidatos a solução, mas através de uma representação mais concisa, como um grafo com pesos (DORIGO; STÜTZLE, 2004).

Tratando do algoritmo em si, a inspiração do ACO vem da busca de alimentos pelas formigas na natureza, que ocorre da seguinte maneira: inicialmente, todas as formigas partem do formigueiro em busca de alimento. Como estas não sabem onde o alimento está, seguem caminhos aleatórios, e conforme caminham, liberam substâncias químicas chamadas feromônios. Quando uma formiga encontra alimento e retorna ao formigueiro, ela marca este caminho com feromônio, podendo variar sua intensidade conforme a quantidade de alimento encontrado, proximidade e outros fatores.

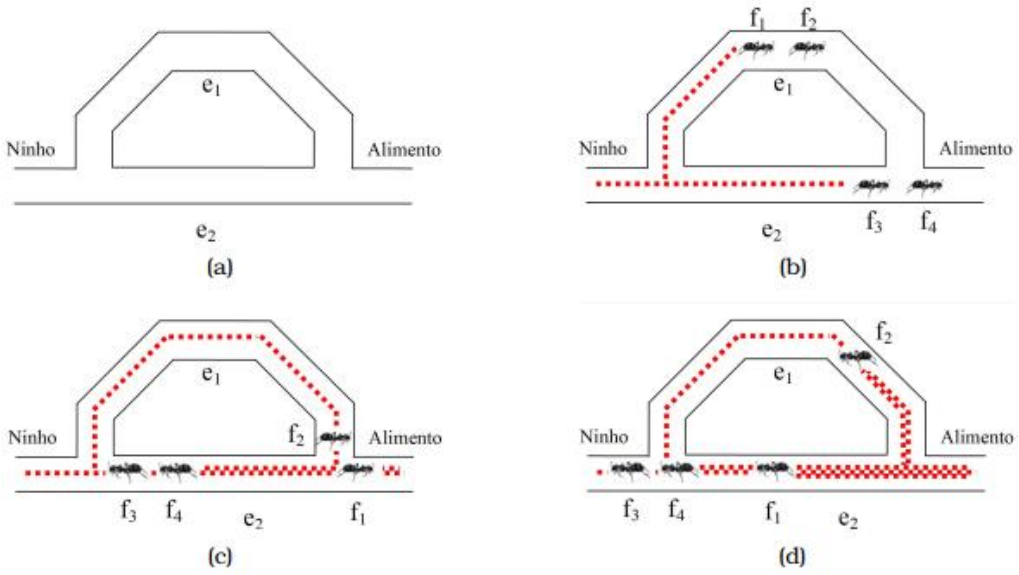
Outras formigas conseguem sentir este feromônio quando passam por locais próximos (é um mecanismo de atuação local) e, da próxima vez que saem do formigueiro, tendem a escolher o caminho do alimento, com maior concentração de feromônio. Desta forma, cada vez mais formigas escolhem este caminho, depositam mais feromônio e o processo se autocataliza. Esse sistema de *feedback* positivo faz com que, após certo tempo, a maioria das formigas tenda a seguir um mesmo caminho (DORIGO; STÜTZLE, 2004).

Goss et al. (1989) comprovou experimentalmente que o caminho encontrado pelas formigas muitas vezes apresenta características ótimas. O exemplo da Figura 2.29 ilustra o experimento do pesquisador utilizando uma população de 4 formigas: tem-se dois caminhos entre o ninho e o alimento, e_1 e e_2, e o primeiro possui o dobro do comprimento do segundo (Figura 2.29.a). Inicialmente, como não há feromônio depositado, as formigas escolhem um caminho aleatório, e supõe-se que metade segue cada caminho (Figura 2.29.b). As formigas f_3 e f_4, que escolheram o caminho mais curto, alcançam o alimento primeiro e retornam ao ninho, depositando mais feromônio no caminho e_2 (Figura 2.29.c). Quando as formigas f_1 e f_2 alcançam o alimento, tem maior probabilidade de escolher o caminho mais curto para voltar (Figura 2.29.d).

Em seguida, o processo se repete (formigas vão buscar o alimento até que essa fonte se esgote), mas agora o caminho e_2 possui mais feromônio depositado, sendo a escolha mais provável para o percurso. Ao fim de certo período, quase todas as formigas optam pelo caminho e_2. Como na natureza o feromônio demora para evaporar, nem todas as formigas escolhem o caminho e_2, o que pode ser enxergado como um mecanismo de exploração local (DORIGO; STÜTZLE, 2004).

O ACO resolve problemas de otimização combinatorial através da simulação desse processo utilizando formigas artificiais, que segundo Santos (2017, p. 36) “tem dois papéis importantes: gerar soluções e atualizar os parâmetros do modelo”. Individualmente, as formigas artificiais apresentam estrutura muito simples, mas graças ao princípio da estigmergia (*stigmergy*), que consiste na capacidade destas se comunicarem indiretamente através de alterações no ambiente (uso do feromônio), o algoritmo permite a solução de problemas complexos em um processo descentralizado de tomada de decisões (DORIGO; STÜTZLE, 2004).

Figura 2.29 – Experimento que comprova a descoberta de caminhos ótimos pelas formigas.



a) Dois caminhos e_1 e e_2 (e_1 tem o dobro do comprimento); b) Primeira busca por alimentos; c) Formigas f_3 e f_4 chegam ao alimento primeiro e retornam; d) Formiga f_1 volta pelo menor caminho (e_2) devido à maior presença de feromônio.

Fonte: Calvo (2012).

Computacionalmente, o algoritmo originalmente proposto tem a seguinte estrutura, usando a aplicação no TSP como exemplo de referência: na etapa de inicialização, o ambiente previamente conhecido (espaço de estados S) é estruturado na forma de um grafo, onde os pesos das arestas representam as distâncias entre

cada vértice. Determina-se então um vértice origem (ninho/formigueiro) e atribui-se uma determinada quantidade inicial de feromônio τ_0 a todas as arestas, permitindo que todas tenham chance de ser escolhidas na primeira execução (assim como no caso real). O conjunto de restrições Ω, para o caso do TSP, é que cada vértice só pode ser visitado uma vez e todos os vértices devem ser visitados, retornando ao inicial.

Em seguida, o algoritmo se inicia e possui três etapas principais (DORIGO; BIRATTARI; STÜTZLE, 2006):

a) Construção das soluções pelas formigas artificiais;
b) Aplicação de um mecanismo de busca local (opcional);
c) Atualização das taxas de feromônio.

A primeira etapa consiste na geração de m formigas artificiais (parâmetro definido pelo usuário), que partem da origem. A probabilidade de uma formiga k se mover do vértice i ao vértice j num dado instante é dada pela equação 2.23.

Nesta equação, τ_{ij} representa a quantidade de feromônio na aresta que leva do vértice i ao vértice j e η_{ij} representa a informação heurística desta mesma aresta. Os parâmetros $\alpha\ (\geq\ 0)$ e $\beta\ (\geq\ 1)$ são definidos pelo usuário e controlam a influência de cada um dos termos na escolha do próximo vértice (feromônio e informação heurística, respectivamente). Os seus valores variam de acordo com o problema abordado. O termo V_i^k representa o subconjunto do espaço S que contém os vizinhos do vértice i que ainda não foram visitados pela formiga artificial k, possuindo n_V termos. Se o vértice j já foi visitado, a probabilidade da formiga se deslocar até ele assume valor nulo (DORIGO; BIRATTARI; STÜTZLE, 2006).

$$P_{ij}^k = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \,.\, \eta_{ij}^{\beta}}{\sum_{l=1}^{n_V} \tau_{il}^{\alpha} \,.\, \eta_{il}^{\beta}}, & se\ j \in V_i^k \\ 0 \quad , & se\ j \notin V_i^k \end{cases} \tag{2.23}$$

A informação heurística, conforme já discutido na seção 2.6.2, agrega conhecimentos específicos do problema ao modelo computacional. No caso do TSP, esta informação geralmente é adotada como o inverso da distância entre os vértices i e j (peso da aresta ij), conforme a equação 2.24. Dessa forma, quanto maior a

distância entre o vértice i (onde a formiga se encontra - atual) e o vértice j (cada opção de deslocamento na iteração atual), menor é a atratividade do deslocamento de i a j.

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{2.24}$$

A primeira etapa ocorre para cada formiga até que todas completem um ciclo hamiltoniano (partida e volta a um mesmo vértice, passando uma única vez por todos os vértices do grafo), que representa um estado objetivo do TSP. Na abordagem do problema das trajetórias do pulso ultrassônico, essa etapa se encerra quando todas as formigas saem da origem e atingem o destino (não há necessidade de retorno e nem todos os vértices precisam ser visitados).

Na segunda etapa do algoritmo, pode ser aplicado um mecanismo de busca local, visando melhorar a qualidade das soluções geradas inicialmente. Apesar de ser opcional e específica para cada problema, esta costuma ser incluída nas implementações mais recentes (DORIGO; BIRATTARI; STÜTZLE, 2006).

Na terceira e última etapa, o feromônio de cada uma das arestas é atualizado de acordo com a equação 2.25 (para a aresta ij).

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\,\tau_{ij}^{k} \tag{2.25}$$

Esse novo valor será adotado na próxima iteração (definida como t+1), com base nas taxas de feromônio atuais (iteração t) subtraídas da taxa de evaporação ρ (valor entre 0 e 1, definido pelo usuário). Por fim, soma-se a influência das novas trilhas de feromônio adicionadas por cada formiga que passou pela aresta ij, de acordo com a equação 2.26.

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k}, & se\ a\ aresta\ ij\ pertence\ ao\ caminho\ da\ formiga\ k \\ 0\ , & caso\ contrário \end{cases} \tag{2.26}$$

Nesta equação, Q é um parâmetro constante (definido também pelo usuário) e L_k diz respeito ao comprimento total do caminho percorrido pela formiga k, que em um problema genérico assume a forma de uma função objetivo a ser minimizada/maximizada. No caso do TSP, formigas que realizarem trajetos melhores (mais curtos ou de menor custo) marcam trilhas mais fortes de feromônio nas arestas, favorecendo a escolha destes caminhos em próximas iterações. O *feedback* positivo proporcional à qualidade do caminho visa aumentar a velocidade de convergência do algoritmo (DORIGO; STÜTZLE, 2004).

Analisando a equação 2.26, é possível concluir que arestas que foram cruzadas por mais de uma formiga, mesmo que estas trilhem caminhos diferentes, poderão receber mais de um depósito de feromônio na mesma iteração, pois apresentam, probabilisticamente, maior potencial na construção de boas soluções.

É importante destacar que o feromônio não é depositado durante o processo de ida das formigas até o alimento, já que nesse momento não é possível medir a qualidade das respostas obtidas (comprimento total do caminho percorrido). O depósito é realizado apenas durante o processo de retorno das formigas ao formigueiro (pelo mesmo caminho de ida), antes do início da próxima iteração (DORIGO; STÜTZLE, 2004).

O algoritmo se repete até que uma das condições seja satisfeita: um determinado número de iterações ou tempo máximo é atingido, a melhor solução atinge um nível de qualidade estipulado ou todas as formigas artificiais passam a seguir o mesmo caminho (SANTOS, 2017).

Desde a sua criação, diversas variações de estrutura têm sido propostas ao ACO, visando melhorar a qualidade das respostas ou a rapidez de convergência do método. Alguns exemplos são: adoção de uma estratégia elitista no *Elitist Ant System* (EAS), semelhante ao que ocorre nos GAs. Nesse caso, a formiga que constrói a melhor solução deposita mais feromônio que as demais (DORIGO; MANIEZZO; COLORNI, 1996). No *MAX-MIN Ant System* (MMAS), apenas as formigas que encontram os melhores caminhos (ao invés de todas) atualizam a taxa de feromônio após cada iteração, fortalecendo a pressão pela escolha destes caminhos (STÜTZLE; HOOS, 2000).

Enquanto isso, na variação *Ant Colony System* (ACS) a regra de escolha do próximo vértice para o qual as formigas irão se deslocar se altera e estas se movimentam paralelamente, atualizando as quantidades de feromônio enquanto

caminham. Além disso, a taxa de atualização do feromônio também é diferente, se baseando na melhor solução encontrada (GAMBARDELLA; DORIGO, 1996). Um resumo das principais variações de estrutura do AS original é apresentado na Tabela 2.3.

Tabela 2.3 – Resumo dos principais algoritmos ACO para problemas NP-difíceis propostos na literatura.

Algoritmo ACO	Ano de desenvolvimento	Aplicado no TSP?
Ant System	1991	Sim
Elitist AS	1992	Sim
Ant-Q	1995	Sim
ACS	1996	Sim
MMAS	1996	Sim
Rank-based AS	1997	Sim
ANTS	1998	Não
Best-Worst AS	2000	Sim
Population-based ACO	2002	Sim
Beam-ACO	2004	Não

Fonte: Adaptado de Gendreau e Potvin (2019).

Comparações entre os diversos algoritmos do ACO indicam que as variantes que possuem melhor desempenho são a MMAS e a ACS, seguidas pelo *Rank-based Ant System* (também chamado de *AS-Rank*), que apresenta resultados levemente inferiores. Por este motivo, as duas primeiras variações são as mais frequentemente utilizadas (DORIGO; STÜTZLE, 2004).

Dorigo e Stützle (2004) resumiram bons valores de parâmetros para a maior parte dos algoritmos de ACO à partir de testes em uma quantidade significativa de instâncias do TSP. Estes valores são apresentados na Tabela 2.4.

Na tabela, n representa o número de vértices do grafo e C^{nn} representa uma estimativa inicial do custo da trajetória entre origem e destino. Para os parâmetros específicos do algoritmo ACS, q_0 e ξ, bons valores encontrados foram 0,9 e 0,1, respectivamente (DORIGO; STÜTZLE, 2004).

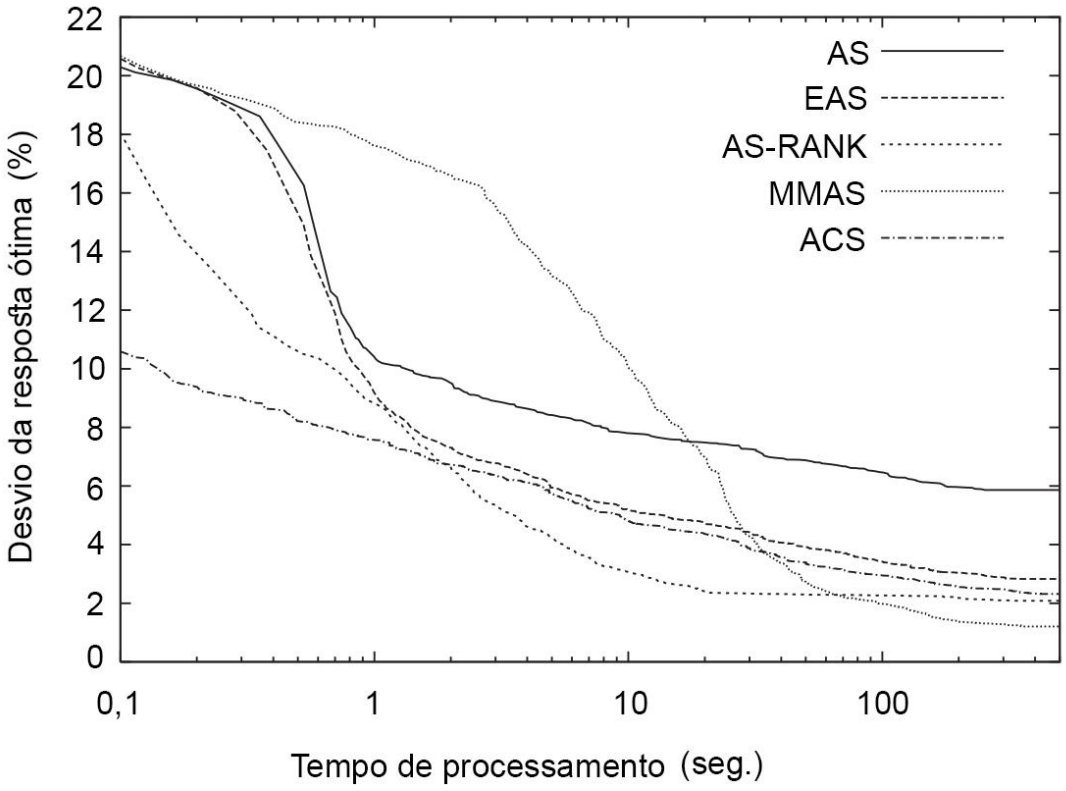
Tabela 2.4 – Bons valores dos parâmetros para algoritmos ACO sem busca local.

Algoritmo ACO	α	β	ρ	m	τ_0
Ant System	1	2 a 5	0,5	n	m/C^{nn}
Elitist AS	1	2 a 5	0,5	n	$(n+m)/\rho.C^{nn}$
ACS	-	2 a 5	0,1	10	$1/n.C^{nn}$
MMAS	1	2 a 5	0,02	n	$1/\rho.C^{nn}$
Rank-based AS	1	2 a 5	0,1	n	$0{,}5.r.(r-1)/\rho.C^{nn}$

Fonte: Adaptado de Dorigo e Stützle (2004).

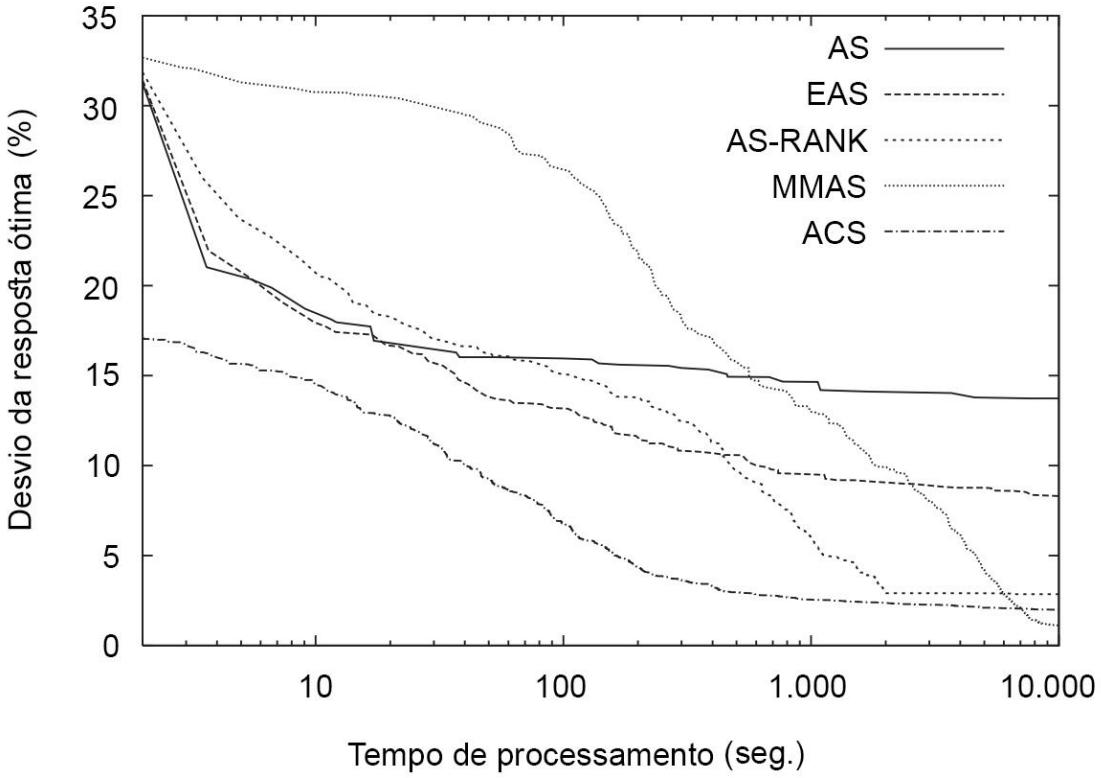
A Figura 2.30 compara algoritmos ACO para a aplicação em uma instância do TSP com 198 cidades, considerando o tempo de processamento (em segundos) no eixo horizontal e a porcentagem de desvio da solução ótima no eixo vertical. A Figura 2.31 compara os mesmos algoritmos para uma instância do TSP maior, com 783 cidades. Em ambos os casos, foram utilizados os valores de referência da Tabela 2.4 para a execução dos algoritmos, com $\beta = 2$.

Figura 2.30 – Comparação entre algoritmos ACO para TSP com 198 cidades.



Fonte: Adaptado de Dorigo e Stützle (2004).

Figura 2.31 – Comparação entre algoritmos ACO para TSP com 783 cidades.



Fonte: Adaptado de Dorigo e Stützle (2004).

Observando as figuras e comparando os algoritmos MMAS e ACS, é possível concluir que o primeiro se inicia com soluções mais distantes da ótima, permanecendo por mais tempo na fase de busca global de soluções (*exploration*), mas atingindo melhores soluções finais quando comparado com os outros algoritmos. Enquanto isso, o ACS parte de soluções inicialmente melhores e explora menos a busca global, passando para a uma busca local (*explotation*) de soluções potencialmente boas em um tempo menor. Isso faz com que boas soluções sejam encontradas mais rapidamente (converge mais rápido), mas que as soluções finais sejam de qualidade levemente inferior. Como o interesse deste trabalho é melhorar o desempenho computacional de algoritmos no que diz respeito a tempo de execução, optou-se pelo uso do ACS.

O processo de inicialização do algoritmo ACS é o mesmo já explorado: num grafo conhecido, atribuem-se à todas arestas uma quantidade inicial de feromônio τ_0 (parâmetro definido pelo usuário), que juntamente com a informação heurística η_0 definem a atratividade de cada vértice do grafo. Em seguida, as m formigas artificiais (parâmetro) iniciam a construção de caminhos do vértice de origem (formigueiro) até

o destino (alimento). A diferença é que agora utiliza-se uma regra pseudoaleatória de escolha, diferente da proposta pela equação 2.23.

Esta nova regra está exemplificada na equação 2.27 para uma formiga que vai do nó i ao j: sorteia-se um número aleatório q. Se este for menor ou igual a q_0 (parâmetro), a formiga escolhe a melhor viagem possível (maior produto feromônio-heurística entre todas arestas do grafo que são vizinhas do vértice atual). Caso contrário, aplica-se uma regra probabilística para a escolha, onde a chance de cada vértice ser escolhido como destino é proporcional ao seu produto feromônio-heurística (regra da roleta da equação 2.23). Caso o vértice j já tenha sido visitado, sua probabilidade de escolha é nula (restrições do problema não se alteram). É possível observar, à partir da equação 2.27, que o parâmetro α, que elevava τ_{ij}, é eliminado na formulação do ACS.

$$j = \begin{cases} \max\left\{\tau_{ij} \,.\, \eta_{ij}^{\beta}\right\} \quad , & se\ q \leq q_0 \\ \\ \dfrac{\tau_{ij} \,.\, \eta_{ij}^{\beta}}{\sum_{l=1}^{n_V} \tau_{il} \,.\, \eta_{il}^{\beta}} \quad , & se\ q > q_0 \end{cases} \tag{2.27}$$

Outra diferença do algoritmo é que as formigas se movimentam paralelamente e, após um passo de todas, a taxa de feromônio nas arestas é atualizada. Essa atualização funciona como um mecanismo de busca local que aperfeiçoa as soluções conforme vão sendo criadas (segunda etapa da aplicação do algoritmo ACO) (DORIGO; STÜTZLE, 2004). No AS original, esta taxa era atualizada quando cada formiga finalizava um caminho completo (da origem ao destino).

A nova atualização dos feromônios funciona da seguinte maneira: após um passo de cada formiga, o feromônio da última aresta que foi atravessada por cada é alterado de acordo com a equação 2.28, que define o valor do feromônio para o próximo passo (DORIGO; BIRATTARI; STÜTZLE, 2006). Na equação, p representa o passo da formiga (diferente da iteração, que representa um ciclo completo) e ξ representa a taxa de evaporação local (parâmetro).

$$\tau_{ij}(p+1) = (1-\xi).\tau_{ij}(p) + \xi.\tau_0 \tag{2.28}$$

À partir da equação 2.28, é possível enxergar que os níveis de feromônio de uma aresta nunca ficam abaixo de τ_0 e uma mesma aresta que foi utilizada por duas ou mais formigas perde mais feromônio que outras. Apesar disso parecer contraditório, pois diminui a quantidade de feromônio de uma aresta que provavelmente é boa, tem o intuito é funcionar como um mecanismo de favorecimento da busca global (*exploration*), evitando que as formigas produzam soluções parecidas (ou até a mesma) e convirjam para um mesmo caminho precocemente.

O processo se repete até que todas as formigas atinjam o destino. Neste momento, calculam-se as distâncias (função objetivo) totais percorridas por cada formiga. A melhor formiga desde o início da execução do algoritmo (chamada de *best-so-far*), ou seja, aquela que percorreu o menor caminho (que possui custo L_{best}), é a única que acrescenta feromônio às arestas que atravessou, vide equação 2.29, aumentando sua atratividade para as formigas da próxima iteração do algoritmo, que se repete até o critério de parada: número de iterações ou tempo atingido, erro estipulado ou estagnação – todas formigas seguindo o mesmo caminho. Na equação, ρ representa a taxa de evaporação global do feromônio (parâmetro).

$$\tau_{ij}(p+1) = (1-\rho).\tau_{ij}(p) + \rho.(L_{best})^{-1} \qquad (2.29)$$

2.8 RESUMO DO CAPÍTULO

Ao longo deste capítulo, o ensaio de ultrassom, que faz parte da classe de ensaios não destrutivos do concreto, foi detalhado. Este consiste em um arranjo de dois transdutores piezoelétricos que emitem e recebem ondas de tensão em altas frequências (acima de 20kHz) dentro de estruturas. O equipamento também mede o tempo de viagem dos pulsos. Como a distância entre os transdutores é conhecida à partir do arranjo experimental, é possível calcular a velocidade do pulso ultrassônico.

Um conjunto de medidas do ensaio pode ser aliado à técnica de tomografia computadorizada, permitindo a geração de tomogramas - mapas de velocidade do pulso ultrassônico em seções transversais da estrutura. Em seguida, o *software* TUSom, que faz uso dessa associação, teve suas principais etapas detalhadas. Estas são: entrada da geometria da seção e de uma malha de discretização; Entrada das coordenadas dos pontos de medição das trajetórias; Definição das linhas de medição destas trajetórias (caminho que o pulso segue dentro da estrutura – etapa que será

alterada em simulação computacional neste trabalho); Entrada de medições experimentais do tempo de viagem do pulso; Cálculo das parcelas da trajetória em cada elemento da malha; Processo de geração de imagens. Explorou-se também a formulação matemática envolvida nas últimas duas etapas do *software*.

Também tratou-se de alguns conceitos importantes de ciência da computação que serão exaustivamente utilizados ao longo do trabalho. O problema das trajetórias se assemelha em vários aspectos ao TSP, um problema de busca clássico do campo. Essa tipologia de problemas está presente em muitos ramos da ciência, sendo separados em duas classes principais: os problemas P e os problemas NP (mais complexos e que não podem ser resolvidos em tempo polinomial). A forma mais comum de representação destes problemas são os grafos, estruturas de dados que usam vértices e arestas com pesos para estabelecer relações entre os dados. Os grafos são computacionalmente implementados através de listas ou matrizes de adjacência.

Por fim, elucidaram-se os algoritmos que serão utilizados na alteração da etapa de mapeamento do pulso ultrassônico no *software* TUSom, sendo integrantes de duas grandes categorias: os algoritmos determinísticos e os algoritmos bioinspirados.

Os determinísticos são representados no trabalho pelos algoritmos de Dijkstra e A*. A sua característica principal é que, dada uma entrada de dados e um conjunto de instruções, estes algoritmos sempre resultam na mesma resposta (saída). Além disso, ambos algoritmos são completos (sempre encontram uma resposta) e ótimos (sempre encontram a melhor resposta possível). Dentre as desvantagens, costumam varrer uma grande parcela do espaço de buscas até encontrar a solução do problema, apresentando desempenho computacional inferior (demandam mais tempo de execução).

Como diferenças entre os determinísticos, o algoritmo de Dijkstra faz parte do grupo de algoritmos de busca não informados, expandindo os vértices do grafo de menor custo conforme vão sendo descobertos. Esse processo se repete até que se encontre o caminho de custo mínimo entre dois vértices: a origem e o destino (selecionados pelo usuário). O algoritmo A* se utiliza de heurísticas, ou seja, informações adicionais e específicas para cada problema, para avaliar a qualidade do estado atual, dado um estado objetivo. Com isso, é possível varrer uma parcela menor do espaço de busca, expandindo apenas vértices que apresentem melhor potencial na construção da solução ótima.

Os algoritmos bioinspirados, por sua vez, são representados pelos algoritmos genéticos e pela otimização por colônia de formigas. Ambos possuem em comum a natureza estocástica, sendo processos que se utilizam da aleatoriedade para guiar uma função objetivo determinada pelo usuário em direção ao seu mínimo/máximo (estado objetivo ótimo de um problema de busca), evitando derivadas complexas e diminuindo a convergência a mínimos locais. Têm potencial para apresentar melhor desempenho computacional e são completos. Apesar disso, não são ótimos (por se basearem na aleatoriedade, nem sempre a melhor solução é encontrada e cada execução gera soluções diferentes).

O algoritmo genético se baseia em conceitos da teoria da evolução, aplicando operadores de seleção, cruzamento, mutação e elitismo (opcional) para atingir a maximização/minimização. Enquanto isso, a otimização por colônia de formigas simula o processo auto-organizado de busca por alimento de algumas espécies de formigas, se apoiando nos mecanismos de estigmergia e *feedback* positivo para se guiar em direção ao máximo/mínimo de uma função.

No capítulo seguinte, a metodologia adotada para a implementação dos algoritmos e para teste em exemplos será detalhada.

3 METODOLOGIA

Relembrando o que já foi exposto na seção 1.1, o objetivo principal deste trabalho é definir a trajetória não linear de pulsos ultrassônicos em estruturas de concreto com mapa de velocidades (tomograma) conhecido. Nesta tarefa, foram utilizados algoritmos determinísticos (Dijkstra e A*) e bioinspirados (GA e ACO, na variação ACS). Estes algoritmos foram implementados no *software* de geração de imagens em estruturas TUSom, desenvolvido em linguagem Pascal pelo Prof. Dr. Vladimir Guilherme Haach, no ambiente de desenvolvimento Lazarus.

Inicialmente, foi feito o estudo teórico dos algoritmos, buscando as estruturas que mais se adequassem ao problema das trajetórias. Em seguida, estes algoritmos foram implementados.

3.1 ALGORITMO DE DIJKSTRA

Como o algoritmo de Dijkstra não possui variações significativas de estrutura, a implementação seguiu os moldes do exposto na seção 2.6.1, chamada de Dijkstra unidirecional. Neste algoritmo, parte-se de uma origem e exploram-se os vértices de custo mínimo até que o destino seja atingido. Origem e destino são pontos de um grafo, definidos pelo usuário.

Uma versão inicial do algoritmo de Dijkstra já havia sido implementada no *software* TUSom, que será chamada à partir de agora de Dijkstra v1. Essa implementação apresentou desempenho muito ruim no que diz respeito ao tempo de processamento, um dos fatores que motivou este trabalho.

A implementação inicial foi reprogramada, com uma série de melhorias:

a) Tentou-se minimizar as linhas de código o máximo possível, tornando o programa mais enxuto e com menos instruções a executar;
b) Foram aplicadas diretivas de otimização do próprio FPC (*Free Pascal Compiler*), o compilador padrão do ambiente de desenvolvimento Lazarus. Mais especificamente, foi utilizado o grupo de otimização O2, que emprega técnicas como otimização aritmética e *loop unroll*, evitando ser muito agressivo, o que poderia causar alterações nos resultados do código;

c) Na versão 1, a montagem do grafo de tempos de viagem entre nós e a execução do algoritmo de Dijkstra eram agrupadas em um único bloco de código, o que fazia com que o grafo completo fosse montado a cada execução do algoritmo. Esse processo é desnecessário, dada a natureza estática do problema das trajetórias, ou seja, após a definição do problema, este permanece o mesmo até o fim da aplicação do algoritmo. Dessa forma, a montagem do grafo pode ser feita uma única vez, economizando tempo. Com isso, parte do tempo de processamento do algoritmo foi trocada por uma etapa de pré-processamento, em que monta-se o grafo antes da sua aplicação;
d) Como o grafo não é direcionado (sem sentido de deslocamento preferencial), a matriz que o representa é simétrica. Dessa forma, calculam-se os tempos de viagem para metade da matriz e para a sua diagonal principal, e os valores restantes podem ser replicados (termo ij é igual ao termo ji). Na primeira versão do programa, a matriz completa era calculada;
e) Na versão inicial (Dijkstra v1), após calcular a distância entre o nó atual e todos os seus vizinhos, estas distâncias eram ordenadas, visando selecionar a menor delas, que indica o próximo nó a ser visitado. Esse processo de ordenação das distâncias tomava muito tempo. A substituição pela função intrínseca "*minvalue*" resolveu este problema, já que somente o nó mais próximo é interesse no momento, sendo desnecessária a ordenação de todas as distâncias;
f) Do ponto de vista do algoritmo, a principal melhoria proposta foi interromper o processo iterativo assim que o nó destino é atingido, visto que a menor distância entre ele e a origem é a única grandeza desejada. Na primeira versão, todo o grafo era varrido, o que nos fornecia a menor distância entre o vértice de origem e todos os outros vértices do grafo, gerando uma série de informações sem utilidade e que aumentavam o tempo de execução do programa.

As alterações descritas geraram a versão Dijkstra v2, com desempenho consideravelmente superior, que será explorado na seção de resultados. Após o algoritmo de Dijkstra, o algoritmo A* foi implementado.

3.2 ALGORITMO A*

O algoritmo determinístico A* também foi implementado nos moldes do apresentado na seção 2.6.2 e a sua implementação partiu do programa Dijkstra v2. Dessa forma, também existe um tempo de pré-processamento envolvido para a montagem do grafo antes da aplicação do algoritmo.

A decisão de partir da implementação de Dijkstra foi tomada porque os algoritmos possuem princípios de funcionamento semelhantes: o custo de cada vértice ($c(V)$ da equação 2.20) é o mesmo para ambos, representando a soma do tempo de viagem dos pulsos ultrassônicos da origem até este vértice. A diferença de A* com relação ao algoritmo de Dijkstra é a existência da informação heurística ($h(V)$ da equação 2.20). A soma do custo com a heurística constitui a função de avaliação do algoritmo, que define a qualidade de cada nó do grafo como candidato à próxima visita. Como deseja-se que o tempo de viagem seja mínimo (o problema das trajetórias é um problema de minimização), o nó que apresentar a menor função de avaliação é o próximo a ser visitado.

A heurística foi adotada como o tempo de viagem em linha reta entre o vértice atual i e o vértice destino D (transdutor receptor), considerando na viagem a maior velocidade nodal presente na seção ($MaxVel$), vide equação 3.2. Nesta equação, h_i representa a heurística do nó i, enquanto x_i e y_i indicam as coordenadas deste nó. x_D e y_D representam a informação análoga para o nó destino (D).

$$h_i = \frac{\sqrt{(x_D - x_i)^2 + (y_D - y_i)^2}}{MaxVel} \tag{3.1}$$

A forma de cálculo da heurística foi escolhida visando garantir que a estimativa do tempo de viagem entre o nó i e o destino D seja sempre menor ou igual ao tempo real de viagem entre eles, o que garante a consistência da heurística (respeita a desigualdade triangular da equação 2.21). Como consequência, a solução encontrada pelo método sempre representa a solução ótima (melhor trajetória possível).

Pelo fato do algoritmo A* requerer a visita de menos estados possíveis do problema de busca (nós do grafo), espera-se que ele apresente um desempenho superior ao do algoritmo de Dijkstra.

3.3 ALGORITMOS GENÉTICOS

Como os algoritmos genéticos apresentam várias estruturas diferentes, sua implementação será melhor detalhada. Assim como os algoritmos anteriores, o grafo é montado em uma etapa de pré-processamento.

Com relação ao algoritmo propriamente dito, Linden (2006) recomenda que a representação dos indivíduos seja sempre a mais natural possível. Por este motivo, cada indivíduo foi representado como um vetor de número inteiros, que representam os números dos nós que serão visitados em uma trajetória. Dessa forma, cada número inteiro representa um gene do indivíduo. Esse vetor possui tamanho n, ou seja, engloba todos os nós pelos quais o pulso pode caminhar até o transdutor destino, partindo sempre do transdutor origem. Isso faz com que todos indivíduos apresentem o mesmo tamanho, facilitando a aplicação dos operadores genéticos. Além disso, os números que compõem cada indivíduo são sorteados aleatoriamente, respeitando um raio de vizinhança de cada nó.

A avaliação de cada indivíduo é dada pelo inverso do tempo de viagem total entre os transdutores origem e destino. Desta forma, transforma-se o problema em um problema de maximização: o indivíduo com menor tempo de viagem possui a maior avaliação, que é o objetivo da busca. Os nós do indivíduo que aparecem após o transdutor destino não contribuem para a sua avaliação. O processo de geração e avaliação de indivíduos é exemplificado na Figura 3.1, para uma malha de elementos com duas divisões na direção horizontal e duas na vertical (comumente chamada de malha 2x2, remetendo à uma matriz), com 10 nós ao todo (9 da malha + 1 gerado pelo transdutor destino). A trajetória que o indivíduo gerado no exemplo representa é marcada com a cor rosa e a trajetória retilínea suposta inicialmente é marcada em azul.

Apesar do modelo de criação de indivíduos adotado gerar caminhos aleatórios inicialmente (respeitam apenas um raio de vizinhança predeterminado), a evolução das gerações se encarrega de eliminar os indivíduos com avaliação ruim. Os indivíduos que apresentam trajetórias retilíneas diretas entre transdutor emissor (origem) e receptor (destino) são penalizados com um aumento de 100 vezes em seu tempo de viagem, visto que já conhecemos estas trajetórias de antemão e a reprodução de indivíduos que as contenham é indesejável.

Figura 3.1 - Exemplo do processo de criação e avaliação de indivíduos no GA.

Fonte: Autor.

O operador de seleção utilizado foi o critério da roleta (equação 2.22), em que a probabilidade de um indivíduo ser selecionado é proporcional à sua avaliação. Como transformamos o problema das trajetórias em um problema de maximização, indivíduos com maior avaliação possuem maiores chances de serem escolhidos para o cruzamento. A escolha deste operador foi feita por sua simplicidade, o que gera rapidez e colabora para um bom desempenho, além da sua grande difusão.

No que diz respeito ao cruzamento, adotou-se o operador de cruzamento uniforme, pelo fato deste tender a apresentar resultados superiores, apesar de ser mais devagar que o cruzamento de dois pontos. A morfologia do cruzamento uniforme foi adaptada para o GA baseado em ordem (*ordering* GA), visto que a ordem em que os nós aparecem na composição de um indivíduo influenciam na sua avaliação (LINDEN, 2006). Ainda deve-se respeitar a restrição de que o pulso não pode passar duas vezes pelo mesmo nó, semelhante ao que ocorre no TSP.

Na aplicação do operador, os pais selecionados são separados em pares, ou seja, dois pais geram dois filhos (população sempre será um número par). Feito isso, gera-se um string de bits (*bitstring*) aleatório, cujos termos podem assumir valores 0 ou 1, determinando quais genes de cada filho gerado provém de qual pai.

Tomando como exemplo o filho 1, quando o *bitstring* assume valor 1, o seu gene provém do pai 1. Quando o *bitstring* assume valor 0, armazenam-se os genes correspondentes do pai 1 em um vetor de “restos”, e ordenam-se estes termos na ordem em que aparecem no pai 2. Após a ordenação, estes genes são alocados no filho 1. O processo é exemplificado na Figura 3.2 para a mesma malha de elementos e nós da Figura 3.1. A aplicação do operador é análoga para filho 2 e pai 2.

Figura 3.2 - Exemplo de aplicação do operador de cruzamento uniforme no GA.

Fonte: Autor.

Para o operador de mutação, novamente utilizaram-se os conceitos do GA baseado em ordem, adotando o operador de mistura de sublistas. Neste operador, quando a mutação ocorre, sorteiam-se dois pontos de corte e misturam-se os genes do indivíduo entre estes pontos, em uma ordem aleatória. Davis[2] (1991, apud LINDEN, 2006, p. 181) afirma que este operador se mostra mais agressivo no aspecto da convergência genética que outros modelos de mutação do GA baseado em ordem, atingindo resultados mais satisfatórios.

Também empregou-se o operador de elitismo, com 2 indivíduos pertencendo à elite por geração. Dessa forma, as duas melhores soluções encontradas desde o início da aplicação do algoritmo (*best-so-far*) são armazenadas na memória e passadas diretamente à próxima geração.

No que diz respeito aos parâmetros do algoritmo, a probabilidade de cruzamento (PC) foi adotada como 0,90 e a probabilidade de mutação (PM) como 0,10, estimulando fortemente as buscas locais. Estes valores são adotados na maioria das publicações e têm atingido bons resultados. Alguns autores afirmam que a soma das porcentagens dos operadores deve ser 1 (LINDEN, 2006).

Os parâmetros do GA que variaram ao longo de sua aplicação nos exemplos estudados foram o tamanho da população e o número de gerações, buscando atingir resultados com uma porcentagem de erro estipulada. Este erro é calculado para cada trajetória de acordo com a equação 3.3, tomando como referência resultados obtidos com a aplicação do algoritmo de Dijkstra, pois este é determinístico e fornece as soluções ótimas. É interessante destacar que ambas versões do algoritmo de Dijkstra (v1 e v2) fornecem as mesmas respostas. O erro médio é obtido pela média do erro de todas as trajetórias do modelo simulado.

[2]DAVIS, L. **Handbook of genetic algorithms.** New York: Van Reinhold Nostrand, 1991.

$$erro = \left(\frac{\Delta t_{GA} - \Delta t_{Dijkstra}}{\Delta t_{Dijkstra}}\right).100 \tag{3.2}$$

3.4 ALGORITMO ACS

O algoritmo ACS foi implementado de acordo com a seção 2.7.2 e, assim como foi feito nos anteriores, existe uma etapa de pré-processamento para montagem do grafo que representa o problema das trajetórias. No programa, tanto a informação heurística (η_{ij}) quanto a quantidade de feromônio (τ_{ij}) de cada aresta do grafo foram armazenadas na forma de matrizes e também tiveram seus valores pré-calculados, visando facilitar a indexação e a chamada de seus valores ao longo da aplicação do algoritmo.

Seguindo a recomendação de Gambardella e Dorigo (1996), utilizaram-se as chamadas *candidate lists* (CL) durante a construção dos caminhos das formigas. As CL armazenam apenas os vizinhos mais próximos de cada nó, evitando que se calcule o produto feromônio-heurística da equação 2.27 para todos os nós do grafo e melhorando o desempenho computacional.

Os valores dos parâmetros (mostrados nas equações 2.27, 2.28 e 2.29) foram adotados conforme recomendado pela Tabela 2.4: $m = 10$, $\beta = 2$, $\xi = \rho = 0{,}1$ (10%) e $\tau_0 = 1/n.C^{nn}$, em que n representa o número de nós do grafo e C^{nn} uma estimativa grosseira da melhor trajetória, adotada como o tempo de viagem em linha reta entre os transdutores de origem e destino do ensaio de ultrassom. Este tempo é calculado à partir da integral de linha da equação 2.10, em que a distância entre os transdutores é conhecida do arranjo experimental (ΔS_i) e calculada de acordo com a equação 2.11, enquanto a expressão do campo de vagarosidades de cada elemento é obtida à partir da resolução do sistema linear da equação 2.12. É importante relembrar que, no presente trabalho, os mapas de velocidade das seções são conhecidos no momento da aplicação dos algoritmos.

O único parâmetro alterado foi q_0, adotado como 0,5 (50%) para favorecer a busca global (*exploration*) e evitar a convergência prematura a mínimos locais. Além disso, outra lógica foi adotada no cálculo da informação heurística, vide equação 3.4: para uma formiga situada no nó i, a atratividade do nó j é dada pelo inverso da soma do tempo de viagem entre os nós i e j (Δt_{ij}) com o tempo de viagem entre o nó j e

transdutor destino (Δt_{jD}). Dessa forma, quanto mais distante o nó j estiver do destino D, menos atrativa é a viagem para este nó, partindo do nó i.

$$\eta_{ij} = \frac{1}{\left(\Delta t_{ij} + \Delta t_{jD}\right)} \tag{3.3}$$

Para aplicação nos exemplos estudados, alteraram-se o tamanho da CL e o número de iterações, buscando atingir resultados com um erro máximo estipulado, calculado conforme a equação 3.5 para cada uma das trajetórias, de forma análoga ao que foi feito no GA. O erro médio, também seguindo o que foi feito no GA, é obtido pela média do erro de todas as trajetórias do modelo simulado.

$$erro = \left(\frac{\Delta t_{ACS} - \Delta t_{Dijkstra}}{\Delta t_{Dijkstra}}\right).100 \tag{3.4}$$

Após a implementação dos algoritmos anteriores, estes foram testados em exemplos de seções com defeitos simulados, visando compará-los em dois aspectos: qualidade das soluções e tempo de processamento. No que diz respeito ao primeiro aspecto, as soluções encontradas pelos algoritmos determinísticos são sempre as mesmas, dada uma mesma entrada de dados, e representam as soluções ótimas. Dessa forma, os tempos de viagem do pulso encontrados por estes algoritmos foram tomados como referência para todas as trajetórias, visando a comparação com as soluções encontradas pelos algoritmos bioinspirados. Como os últimos possuem natureza probabilística, variam a cada execução e nem sempre encontram a solução ótima.

No que diz respeito ao segundo aspecto (tempo de processamento), os algoritmos serão executados e terão seus tempos registrados, utilizando uma mesma máquina e as mesmas condições para todos. Para os algoritmos que apresentam tempo de pré-processamento embutido (todos, exceto Dijkstra v1), este tempo estará incluído no tempo total do algoritmo. No entanto, antes de apresentar os exemplos em que os algoritmos foram testados, serão tratados alguns aspectos do software TUSom.

3.5 SOFTWARE TUSOM

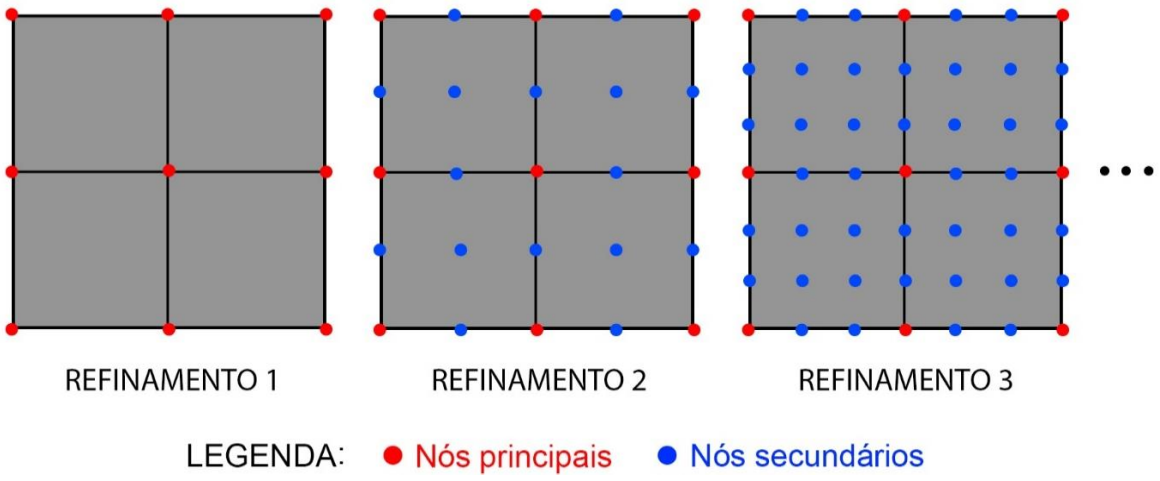
Computacionalmente, é importante salientar que todos os algoritmos anteriores foram implementados em códigos sequenciais, ou seja, não paralelizados (são executados por apenas um núcleo de processamento). Essa é uma das sugestões para desenvolvimentos futuros, tratada posteriormente neste trabalho.

Conforme explorado na seção 2.4, em sua última etapa, o *software* TUSom calcula as velocidades do pulso ultrassônico nos nós de uma seção transversal da estrutura. Estes nós são pontos de interseção entre as linhas de uma malha de elementos definida pelo usuário, exemplificada para uma malha 2x2 na Figura 3.3.

Os pontos em vermelho serão chamados nós principais. Além dos principais, existe no *software* a opção de aumentar o refinamento da malha de nós, fazendo surgir os chamados nós secundários. Nestes, a velocidade é definida à partir da interpolação linear das velocidades nos nós principais. Ao todo, são 10 níveis diferentes de refinamento da malha de nós.

Observa-se na Figura 3.3 que houve um refinamento da malha de nós, mas a malha de elementos permaneceu a mesma. Esse processo será adotado na execução dos dois exemplos testados: inicialmente se define uma malha de elementos, cuja interseção das linhas gera os nós principais. A malha de nós será refinada, mas a malha de elementos não se altera ao longo de toda aplicação dos algoritmos, ou seja, o número de elementos nunca muda. É importante ressaltar a diferença entre malha de elementos e malha de nós neste momento, de forma a evitar confusão posterior.

Figura 3.3 - Exemplo de malha de elementos 2x2 e níveis de refinamento da malha de nós.



Fonte: Adaptado de Giglio e Haach (2020b).

Na aplicação dos algoritmos, é por estes nós que o pulso irá caminhar entre os transdutores do ensaio, tomados como pontos de origem e destino. Quanto maior o refinamento da malha de nós, maior é a quantidade destes e mais fiel à realidade se torna a trajetória seguida pelos pulsos ultrassônicos. Computacionalmente, as coordenadas dos nós foram armazenadas em uma matriz de dimensões n x 2, em que n representa o número total de nós da malha. Dessa forma, cada linha da matriz representa um nó e a primeira e segunda colunas representam suas coordenadas x e y, respectivamente.

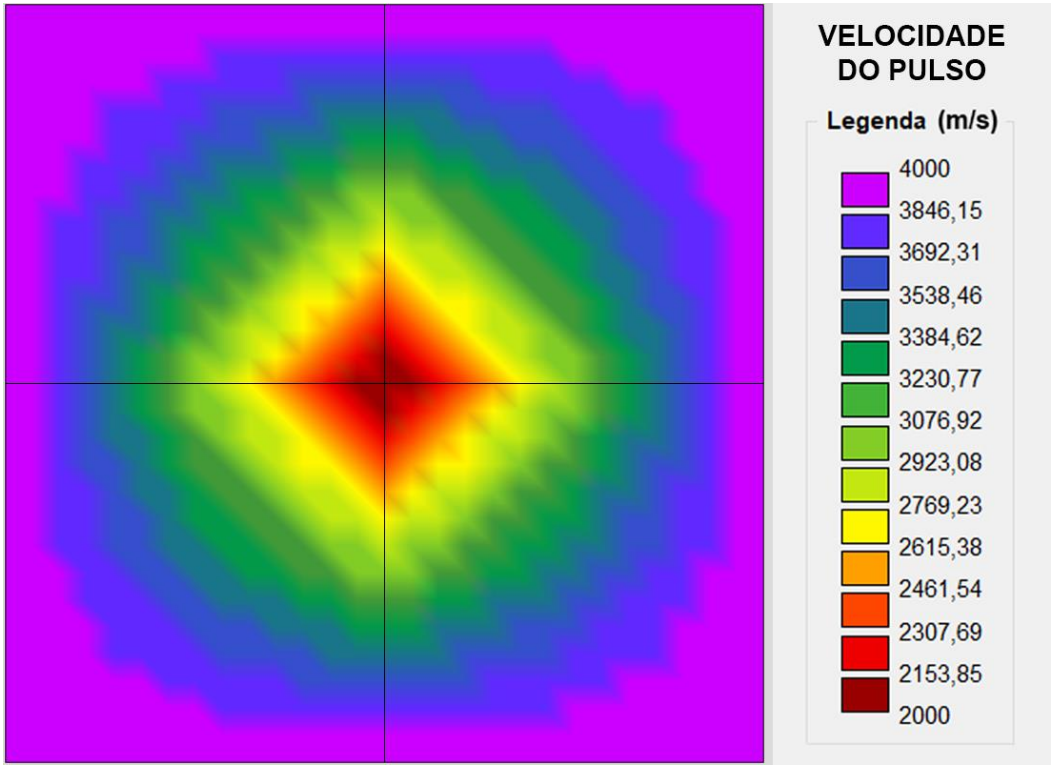
A malha de nós foi estruturada na forma de um grafo não dirigido, ou seja, que não possui sentido preferencial de deslocamento (deslocamento do nó i para o nó j tem o mesmo custo que o deslocamento de j para i). Neste grafo, todos os nós se conectam e o peso das arestas representa o tempo de viagem do pulso entre nós (não mais a distância, como ocorria no TSP).

Dessa forma, não existem pesos negativos (o tempo de viagem é sempre maior que 0), o que respeita a restrição de aplicação do algoritmo de Dijkstra. Como o grafo é denso (todos os nós se conectam), optou-se pela sua representação computacional na forma de uma matriz de adjacência de dimensões n x n, já que ela apresenta maior facilidade de indexação, apesar do maior consumo de memória. Conforme aumenta o número de nós da malha, o tamanho do espaço de buscas também cresce, exigindo maior tempo de processamento para o encontro de soluções ótimas, conforme visto na seção 2.5.

Após a definição da malha de nós e do seu nível de refinamento, que implica no número de nós na seção, define-se uma velocidade em cada um dos nós principais da malha, visando simular defeitos em seções de concreto. Tomando como ponto de partida os valores da Tabela 2.1, o concreto é assumido como de boa qualidade nas regiões de velocidade mais alta. Em regiões de velocidade menor, supõe-se que existem danos ou defeitos de concretagem.

Para a seção da Figura 3.3 com o nível 1 de refinamento da malha de nós (9 nós no total), por exemplo, é possível simular um grande defeito central adotando velocidade de 2000m/s no nó central e de 4000m/s nos nós restantes, resultando no campo de velocidades da Figura 3.4. Como consequência da interpolação linear das velocidades nodais em cada um dos elementos, o dano se estende por grande parte da área destes, que acaba por apresentar velocidades intermediárias, entre a máxima e a mínima.

Figura 3.4 - Exemplo de campo de velocidades para uma seção transversal quadrada.



Fonte: Autor.

No presente trabalho, o interesse principal não é o de calcular campos de velocidade em seções transversais de concreto (tomogramas), função já consolidada do *software* TUSom. Pelo contrário, deseja-se, à partir de campos de velocidade preestabelecidos, simular defeitos em seções de concreto simples (sem armadura) e utilizar os algoritmos determinísticos e bioinspirados para mapear a trajetória que seria seguida pelos pulsos ultrassônicos na realidade, obtendo o tempo total de viagem do pulso nesta trajetória. Ou seja, conhecendo o campo de velocidades da seção transversal, simula-se o ensaio de ultrassom computacionalmente, obtendo como resultado tempos de viagem de cada trajetória e o caminho percorrido por elas.

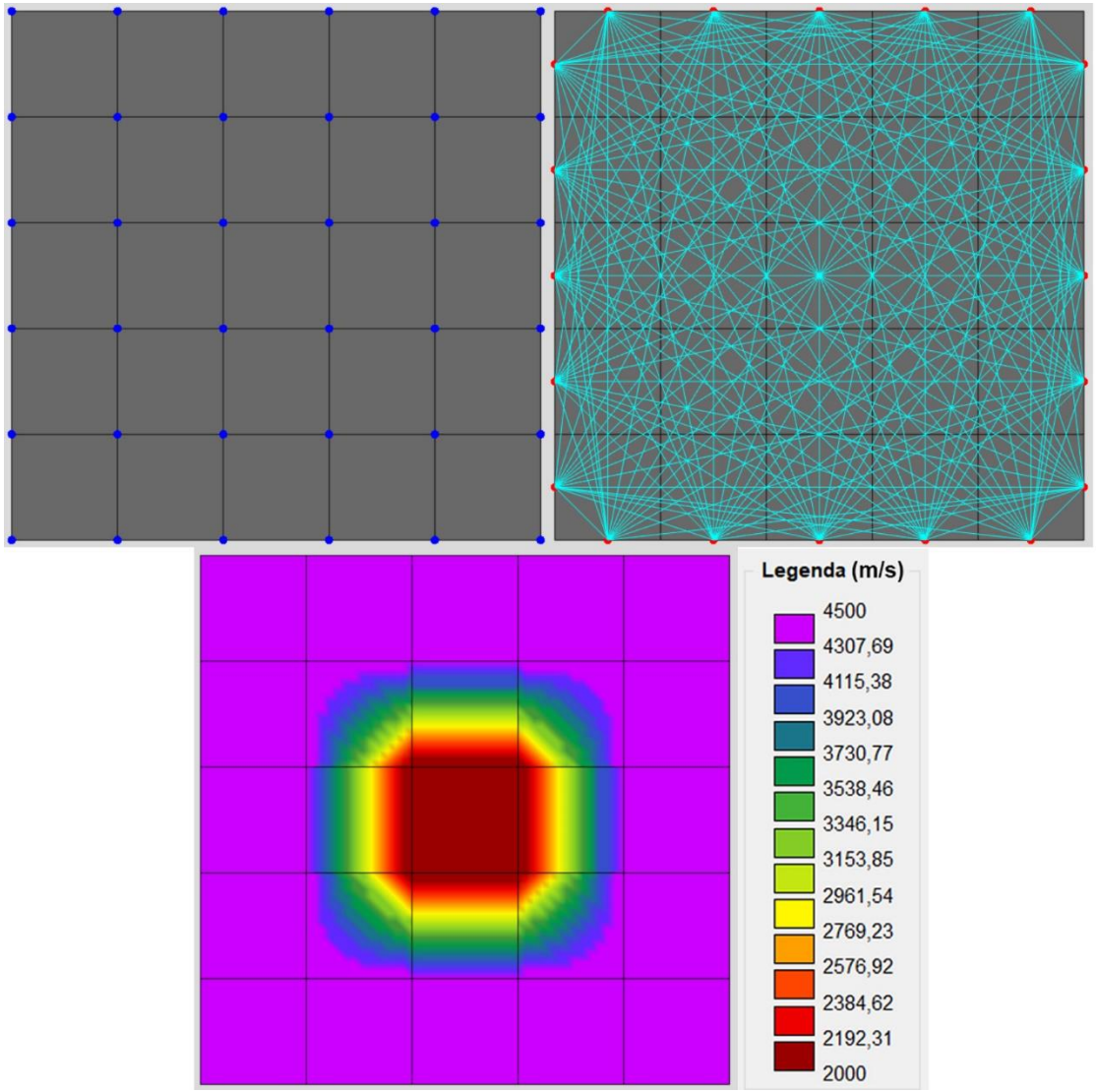
Esse processo é realizado com o objetivo de abandonar a hipótese de que as trajetórias seguidas pelos pulsos no interior dos materiais são retilíneas no ensaio de ultrassom. Esta hipótese é, na maioria das vezes, um comportamento distante do real e que gera distorções no processo de geração de imagens tomográficas de estruturas.

Após esclarecer estes pontos, é possível tratar dos exemplos criados para testar os algoritmos, implementados no *software* TUSom.

3.6 EXEMPLOS SIMULADOS

Os exemplos elaborados para o teste dos algoritmos foram duas seções transversais de concreto simples (sem armadura). A primeira é quadrada, possui dimensões de 500x500mm e foi dividida em uma malha de 5x5 elementos (25 elementos ao todo). Desta forma, cada elemento possui dimensões 100x100mm. Os nós principais são 36, representados como pontos azul-escuros na Figura 3.5. Como descrito anteriormente, esta malha de nós será refinada, aumentando o número de nós e consequentemente simulando trajetórias mais próximas das reais. No entanto, a malha de elementos permanece constante ao longo de todo o processo de aplicação dos algoritmos.

Figura 3.5 – Exemplo 1 (seção quadrada) utilizada no mapeamento do pulso ultrassônico.

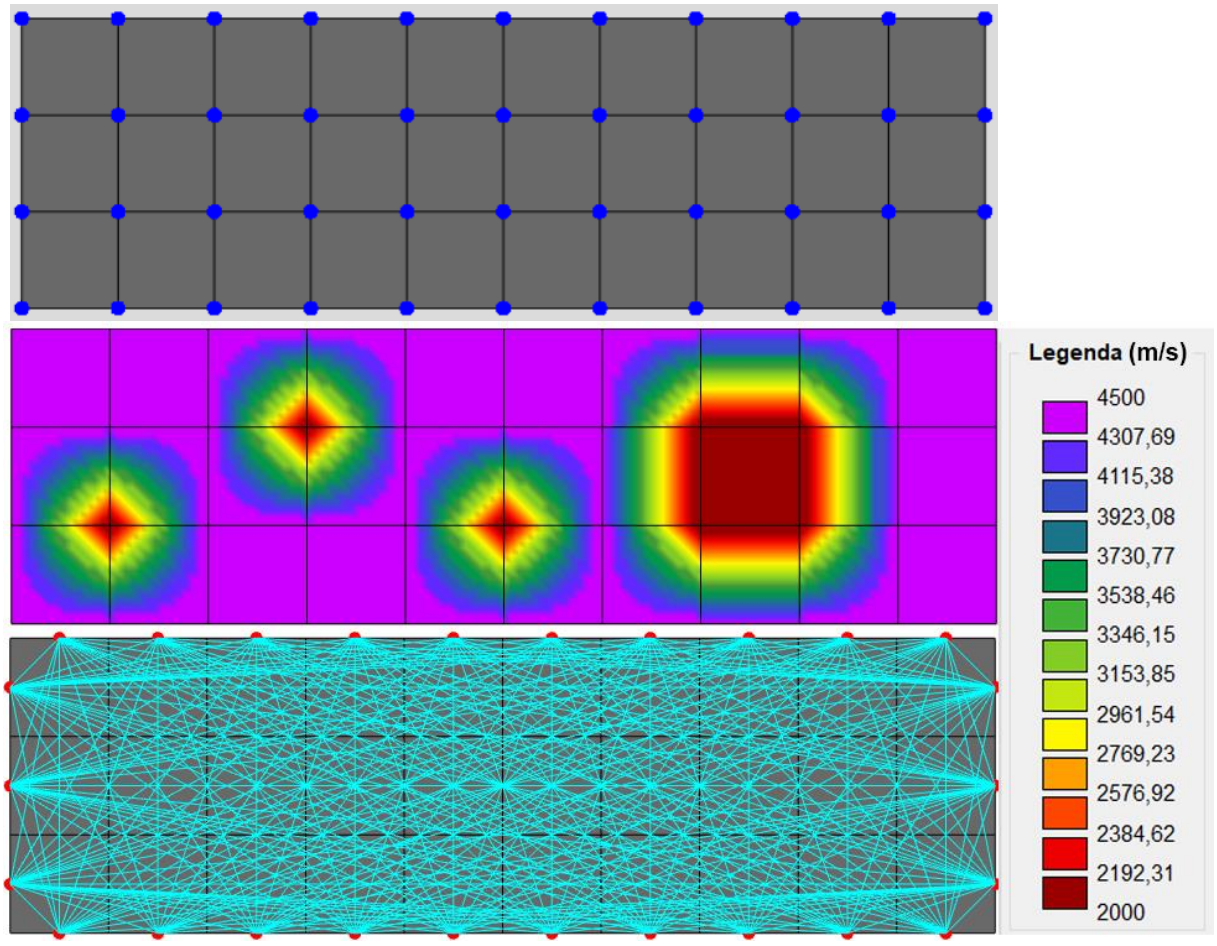


Fonte: Autor.

Nesta seção, simulou-se um defeito central de dimensões 100x100mm (1 elemento completo) adotando 2000m/s de velocidade nos 4 nós centrais e 4500m/s no restante dos nós da malha, representando um concreto íntegro. O defeito acaba se estendendo parcialmente a outros elementos pela interpolação linear de velocidades adotada na formulação, já explorada na seção 2.4. Além da malha adotada e dos defeitos preestabelecidos, a figura também exibe os transdutores do ensaio de ultrassom (pontos vermelhos) e as trajetórias retilíneas supostas inicialmente (150 ao todo).

A segunda seção possui dimensões 150x500mm, representando um corte longitudinal do corpo de prova padronizado do ensaio de flexão (ABNT, 2016). A seção foi discretizada em uma malha de 3x10 elementos quadrados (30 elementos ao todo), com 44 nós principais (Figura 3.6). Cada elemento possui dimensões 50x50mm.

Figura 3.6 - Exemplo 2 (seção retangular) utilizada no mapeamento do pulso ultrassônico.



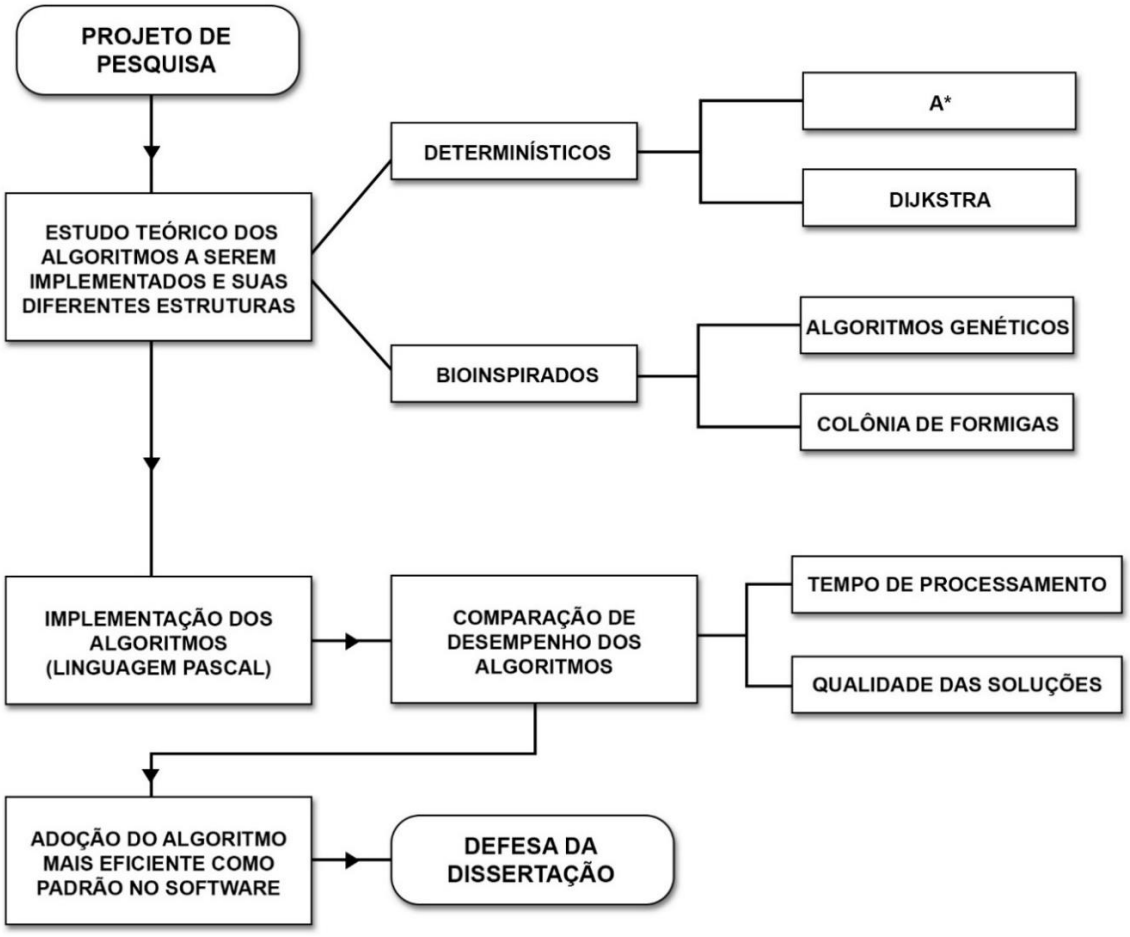
Fonte: Adaptado de Giglio e Haach (2020a).

Nesta seção, buscou-se simular uma situação mais complexa, com mais defeitos e arranjo sem simetria. Ao todo, três defeitos pontuais e um defeito grande foram simulados na seção da mesma forma que na anterior, ou seja, adotando velocidades de 2000 m/s nos nós onde se encontram, enquanto o restante da malha possui velocidade de 4500 m/s, representando um concreto de boa qualidade. Assim como na Figura 3.5, as trajetórias retilíneas supostas inicialmente são exibidas, sendo 229 ao todo.

A aplicação dos algoritmos visa mapear caminhos do pulso ultrassônico mais próximos dos reais, visto que o pulso segue os caminhos de maior velocidade, que correspondem à parcelas mais homogêneas do material. O processo é semelhante ao que ocorre com a eletricidade, que percorre os caminhos de menor resistência.

Ao final da aplicação dos algoritmos nos exemplos, a melhor alternativa será adotada como padrão na etapa de mapeamento de trajetórias do *software* TUSom. A Figura 3.7 apresenta um esquema que reúne as etapas desenvolvidas no trabalho.

Figura 3.7 - Esquema com as etapas do trabalho de mestrado.



Fonte: Autor.

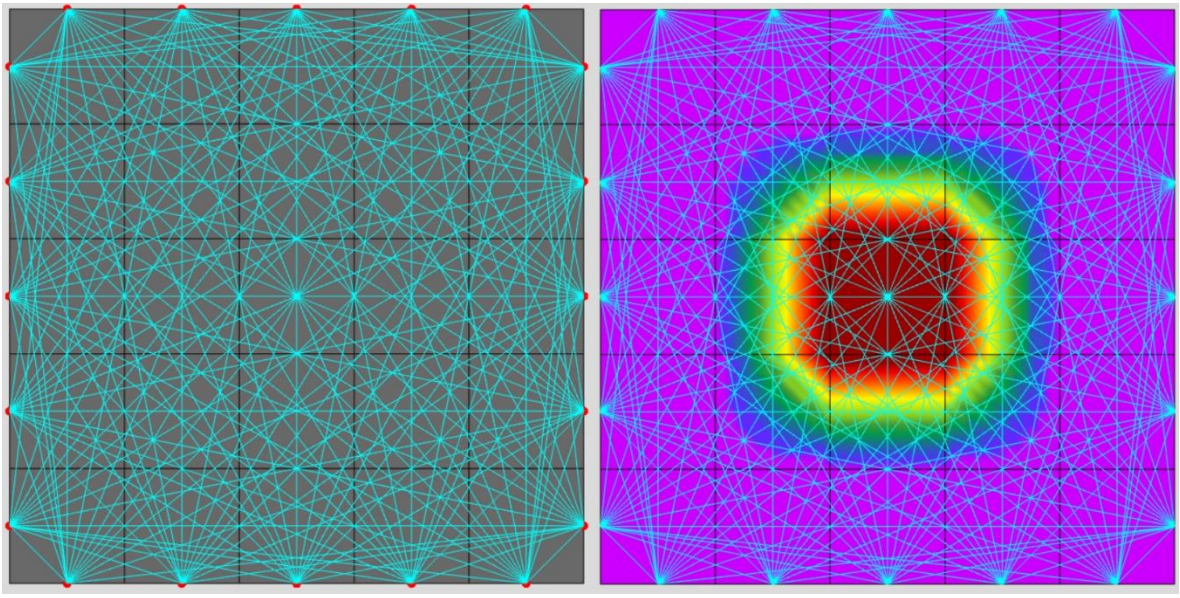
4 RESULTADOS E DISCUSSÃO

Para melhor organização do capítulo, os resultados obtidos em cada um dos exemplos serão analisados inicialmente em seções separadas. No capítulo referente às conclusões, estes resultados serão reunidos e, quando possível, extrapolados.

4.1 EXEMPLO 1

Visando comparar as trajetórias obtidas após o mapeamento e as trajetórias retilíneas supostas inicialmente, sobrepuseram-se as imagens do campo de velocidades e das trajetórias retilíneas da Figura 3.5, obtendo a Figura 4.1. A partir dela, é possível enxergar que os pulsos ultrassônicos atravessam regiões de baixa velocidade do material, onde o defeito se encontra. Isso se traduz em um comportamento distante do real, já que é conhecido que os pulsos atravessam o material pelos pontos de maior homogeneidade, que consequentemente geram a maior velocidade de propagação.

Figura 4.1 – Sobreposição do campo de velocidades e das trajetórias retilíneas no exemplo 1 (seção quadrada).



Fonte: Autor.

Por si só, a consciência de um comportamento distante do real é suficiente para motivar estratégias de mapeamento, que irão se traduzir em melhoria na interpretação

dos resultados do ensaio de ultrassom, na geração de imagens tomográficas e na simulação computacional do ensaio. Analisando ainda alguns dados sobre a seção, presentes na Tabela 4.1, temos que a velocidade máxima na seção é de 4500m/s e a mínima é de 2000m/s. Isso representa uma diferença de 2500m/s, ou seja, a menor velocidade presente na seção é cerca de 55,56% menor que a maior velocidade. Jackson e Tweeton (1994) afirmam que a adoção de trajetórias retilíneas não implica em grandes prejuízos aos resultados de tempo de propagação dos pulsos quando a diferença de velocidades na seção não ultrapassa 10%. Dessa forma, a diferença presente na seção é muito superior à recomendada pelos autores, mais um fator que motiva a adoção de estratégias de mapeamento.

Tabela 4.1 – Informações sobre o exemplo 1 considerando trajetórias retilíneas.

Informações do exemplo 1 - Trajetórias retilíneas	
Número de trajetórias	150
Velocidade máxima na seção	4500m/s
Velocidade mínima na seção	2000m/s
Área da seção com defeitos	33%
Área da seção íntegra	67%
Erro médio	9,31%
Erro máximo	31,42%
Erro mínimo	0,00%
Nº de trajetórias com erro mínimo	60 (40% do total)
Desvio padrão	11,45%

Fonte: Autor.

Os algoritmos determinísticos retornam os menores tempos de viagem possíveis para dada malha de nós, que não necessariamente coincidem com os resultados que seriam obtidos no ensaio real, já que isso só ocorreria para uma malha com número de nós tendendo ao infinito, que simularia um meio contínuo.

Quando comparamos o tempo de propagação dos pulsos ultrassônicos na seção utilizando a hipótese de propagação retilínea e os algoritmos determinísticos, na malha com o maior nível de refinamento de nós, observa-se um erro médio de 9,31% considerando as 150 trajetórias do exemplo, um valor não desprezível. Apesar disso, este erro é menor que o esperado, visto que cerca de 33% da área da seção

apresenta algum tipo de dano, danos estes bem acentuados na região central da seção (menos da metade da velocidade máxima).

Outro indício de que o mapeamento é necessário é o erro máximo obtido na seção, de 31,42%. Este valor é considerável e maior que o triplo do erro médio. Em termos práticos, isso significa que a pior trajetória retilínea possível, que atravessa completamente a região danificada, é 31,42% mais lenta que a melhor trajetória, que parte dos mesmos pontos de origem e destino. Apesar disso, este erro é quase metade da diferença de velocidades nas áreas íntegra e mais danificada da seção, de 55,56%.

O erro mínimo obtido na comparação foi de 0,00%, indicando que, em algumas trajetórias, a aproximação retilínea coincide com a trajetória ótima encontrada pelos algoritmos determinísticos. Do total de 150 trajetórias, este erro foi encontrado em 60 (40%), um número considerável. Esse dado provavelmente deve-se ao fato de que 67% da seção não possui danos, o que faz com que muitas trajetórias retilíneas sejam as mais rápidas entre os transdutores. Estas trajetórias também contribuem para a diminuição do erro médio, podendo dar uma falsa impressão de que este é baixo, hipótese reforçada pelo erro máximo alto, de 31,42%.

Uma grande quantidade de trajetórias retilíneas ótimas não representa o comportamento esperado num arranjo experimental real, já que o concreto, por mais bem dosado que seja, é um compósito. Isso implica que ele é naturalmente heterogêneo (possui trechos com agregado graúdo, trechos de material cimentício e pequenos vazios) e dessa forma é muito pouco provável que um pulso siga uma trajetória totalmente retilínea em seu interior.

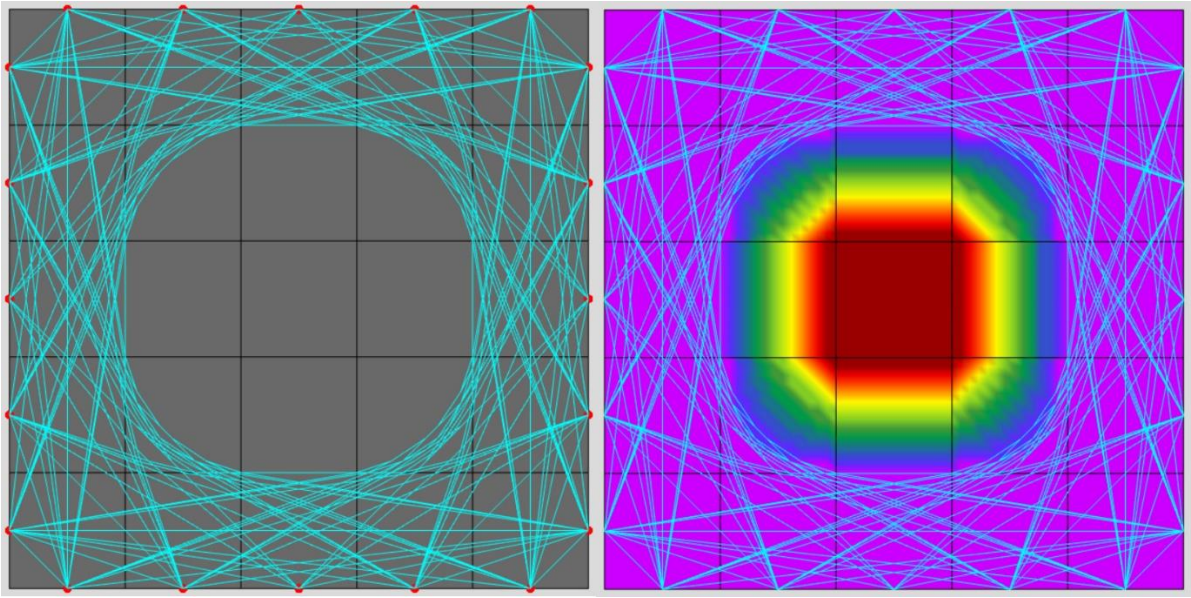
Após o mapeamento das trajetórias utilizando cada um dos algoritmos propostos no exemplo 1 (seção quadrada), para a malha de nós mais refinada, o resultado visual (qualitativo) é apresentado nas Figuras 4.2, 4.3 e 4.4. É importante salientar que, tomando como limite superior o erro médio de 9,31% gerado pelas trajetórias retilíneas quando comparadas às mapeadas, proveniente da Tabela 4.1, trabalhou-se com uma tolerância de erro máxima no GA e no ACS, já que estes são métodos de natureza probabilística, que nem sempre encontram as soluções ótimas, podendo conter erros.

O erro máximo do GA foi adotado com valor de 5%, buscando-se uma diminuição de aproximadamente 5% do erro gerado pela hipótese de trajetórias retilíneas, ou seja, buscam-se soluções com qualidade superior. Enquanto isso, o erro

obtido pelo ACS já foi consideravelmente menor com a quantidade mínima de vizinhos e iterações (parâmetros variáveis do modelo), sendo por este motivo fixado em 0,5%. No entanto, procurou-se manter os algoritmos competitivos com os determinísticos no que diz respeito ao tempo de processamento.

A Figura 4.2 apresenta o resultado do mapeamento do pulso ultrassônico utilizando os algoritmos determinísticos (Dijkstra e A*). Como ambos encontram a solução ótima dos tempos de viagem do pulso, o resultado dos algoritmos é o mesmo e eles serão comparados apenas quantitativamente, através do tempo de processamento.

Figura 4.2 – Resultado do mapeamento do pulso ultrassônico no exemplo 1 (seção quadrada) para os algoritmos de Dijkstra e A* (determinísticos).



Fonte: Autor.

Observando a figura, é possível concluir que o mapeamento foi bem-sucedido, pois é muito clara a tendência dos pulsos desviarem do defeito simulado, buscando o material homogêneo, que fornece maior velocidade e consequentemente o menor tempo de viagem. Esse comportamento é mais fiel ao real e era o resultado esperado. É visível também a grande concentração de trajetórias em pequenos caminhos preferenciais, no limite da área que conservou o concreto íntegro.

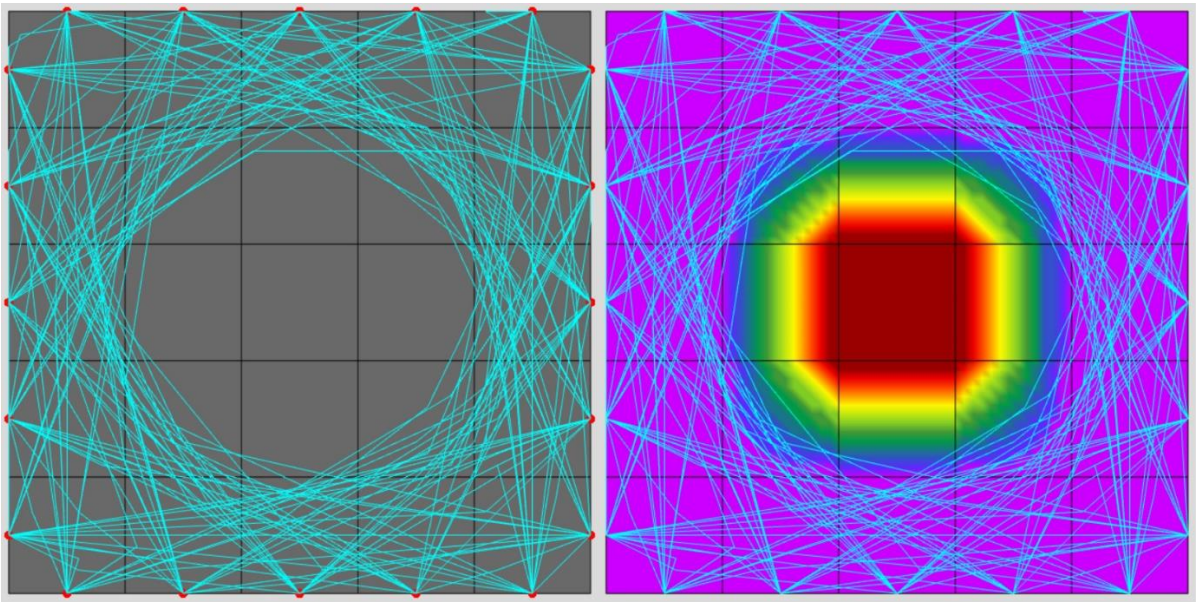
Como mostrado pela Tabela 4.1, a porcentagem de área com algum tipo de dano na seção é de cerca de 33%, mais que apenas o elemento central devido à interpolação linear das velocidades nos elementos. Como consequência, algumas das

trajetórias coincidem em trechos próximos ao dano, buscando desviar deste. Em alguns casos, estas chegam a se confundir e, na totalidade da imagem, aparenta-se ter menos trajetórias que na Figura 4.1. Nas parcelas do material em que não houve dano, é possível observar que as trajetórias permaneceram retilíneas.

Na Figura 4.3, exibe-se o resultado obtido após a aplicação do GA. Comparando-a com a Figura 4.2, é nítida a presença de menos trajetórias coincidentes. Por este motivo, a seção aparenta ter mais trajetórias ao todo. Isso se deve principalmente à natureza probabilística do GA, que faz com que cada trajetória seja um problema de busca individual, guiado até seu mínimo por operadores que envolvem aleatoriedade. Com isso, as respostas se alteram a cada execução e é rara a obtenção de trajetórias coincidentes para dois pulsos diferentes.

Além disso, existe uma tolerância de 5% de erro atrelada ao algoritmo, o que faz com que as soluções encontradas sejam subótimas (respostas aproximadas), em prol de um tempo de processamento menor. Caso o GA fosse executado com mais indivíduos e por mais gerações (números explorados posteriormente), encontraria trajetórias mais próximas das exibidas na Figura 4.2.

Figura 4.3 – Resultado do mapeamento do pulso ultrassônico no exemplo 1 (seção quadrada) para o algoritmo genético (GA).



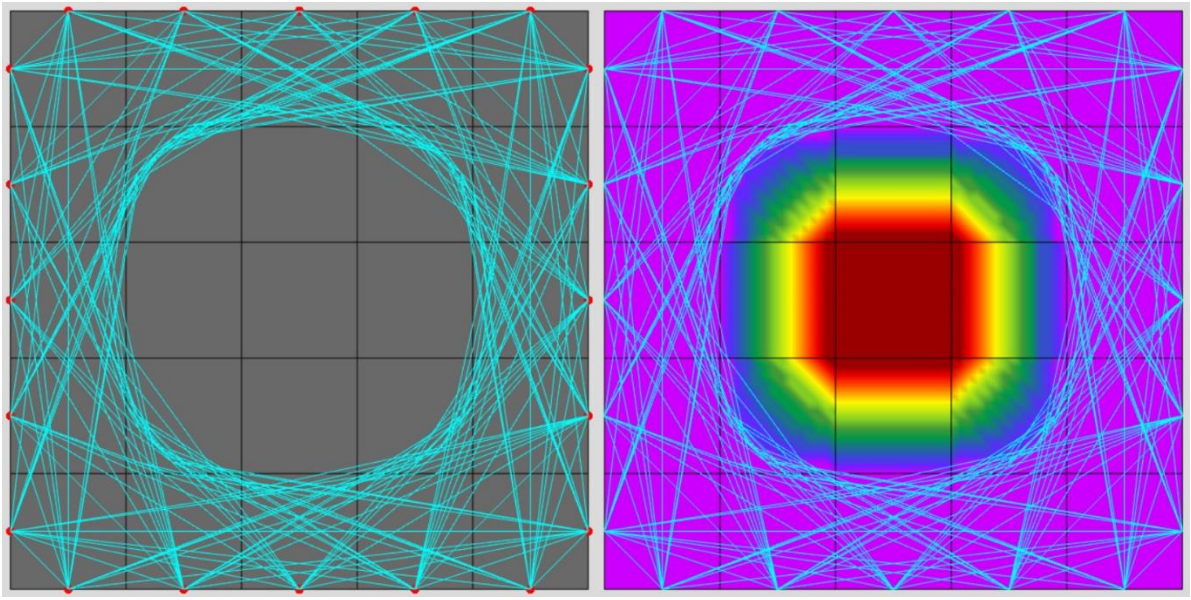
Fonte: Autor.

De fato, observa-se na Figura 4.3 que, para algumas trajetórias, o GA não foi capaz de encontrar as melhores soluções. Essa conclusão é obtida a partir de

trajetórias ruins vistas na figura, como trajetórias que atravessam regiões levemente danificadas (cores azul-escuro e verde na figura). Como o método é probabilístico, não existe garantia de obtenção das melhores soluções e, em alguns casos, os resultados obtidos são ruins. Apesar disso, devido à tolerância baixa de erros (apenas 5%), nenhuma trajetória apresentou um traçado muito ruim, atravessando a região principal do dano, por exemplo (áreas amarelas e vermelhas da Figura 4.3).

A Figura 4.4 apresenta o resultado do mapeamento após a aplicação do ACS. Quantitativamente, os resultados são consideravelmente diferentes com relação às trajetórias da Figura 4.3, visto que o GA opera com uma tolerância máxima de erro de 5%, enquanto a tolerância de erro do ACS é de apenas 0,5%. No entanto, qualitativamente os resultados das figuras são semelhantes.

Figura 4.4 – Resultado do mapeamento do pulso ultrassônico no exemplo 1 (seção quadrada) para o algoritmo *Ant Colony System* (ACS).



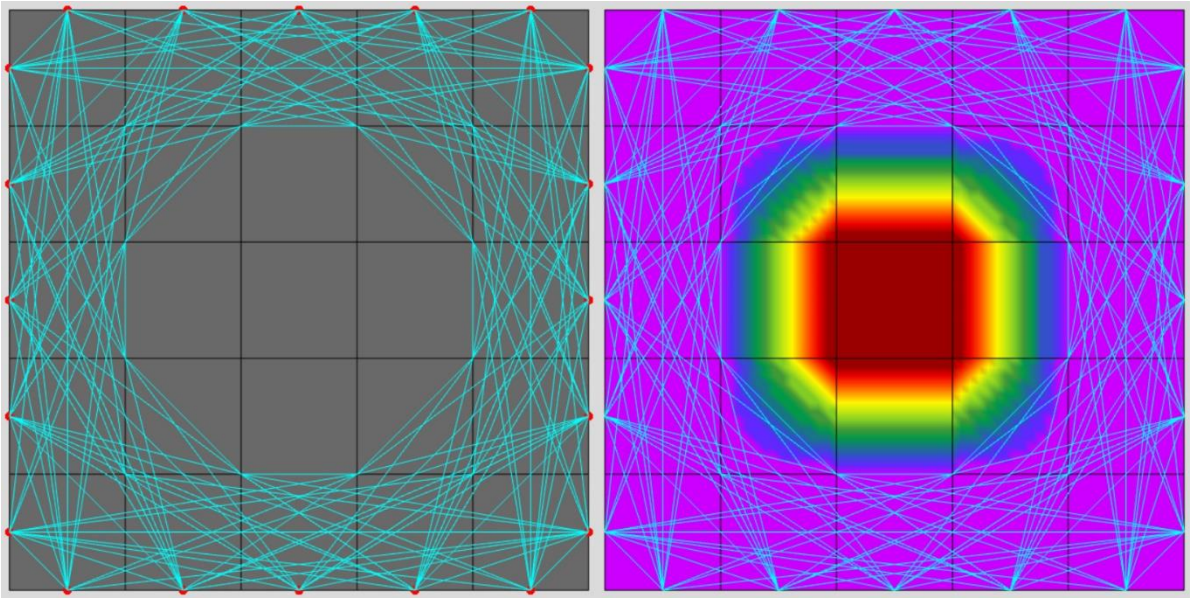
Fonte: Autor.

Pela tolerância menor de erros, a Figura 4.4 se assemelha mais ao resultado encontrado na aplicação dos algoritmos determinísticos e existem mais sobreposições de trajetórias. No entanto, a natureza probabilística do algoritmo ainda se faz presente, sendo visível a obtenção de algumas trajetórias subótimas que atravessam regiões levemente danificadas (em azul-escuro na figura). Diferente do observado no GA, regiões mais danificadas, em verde, não foram cruzadas pelos pulsos ultrassônicos.

Ainda que as respostas obtidas pelos algoritmos bioinspirados sejam visivelmente diferentes das obtidas pelos determinísticos, encontram-se dentro do esperado e considera-se que o mapeamento do pulso também foi bem-sucedido para estes algoritmos.

Outro ponto interessante é que, mesmo para malhas de nós não muito refinadas, as estratégias de mapeamento geram um aumento na qualidade dos resultados muito grande e que compensa o esforço computacional despendido nessa tarefa. Um exemplo disso é o mapeamento do pulso aplicando o algoritmo de Dijkstra à malha de nós com refinamento mínimo do exemplo, que possui apenas 36 nós principais (Figura 4.5).

Figura 4.5 - Resultado do mapeamento do pulso ultrassônico no exemplo 1 (seção quadrada) usando o algoritmo de Dijkstra na malha menos refinada (36 nós).



Fonte: Autor.

A Tabela 4.2 apresenta os erros obtidos com a utilização desta alternativa. Observa-se um erro médio de apenas 0,79% quando os tempos de viagem são comparados com os obtidos na aplicação do mesmo algoritmo na malha de nós com maior refinamento (2601 nós). O erro máximo obtido em uma única trajetória também foi notavelmente baixo, de 2,45%, enquanto o erro mínimo foi de 0,00%, encontrado em 60 trajetórias. Isso indica que, em algumas trajetórias, o caminho mais rápido coincide para o menor e o maior nível de refinamento da malha de nós.

Tabela 4.2 – Erros no exemplo 1 considerando mapeamento com o algoritmo de Dijkstra na malha menos refinada (36 nós).

Erros no exemplo 1 - Algoritmo de Dijkstra	
Número de trajetórias	150
Número de nós da malha	36 (mínimo)
Erro médio	0,79%
Erro máximo	2,45%
Erro mínimo	0,00%
Nº de trajetórias com erro mínimo	60 (40% do total)
Desvio padrão	0,82%

Fonte: Autor.

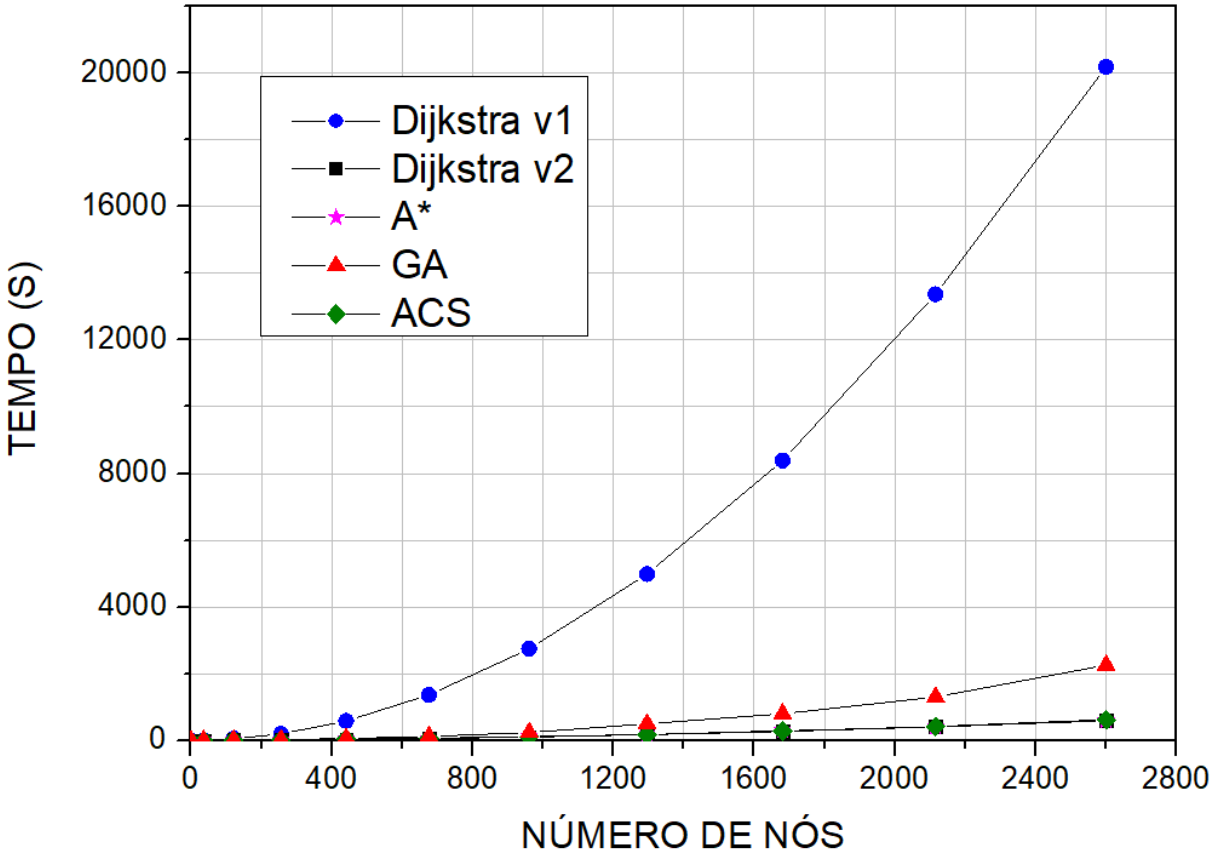
Nota-se que o número de trajetórias ótimas (60) encontrada com a aplicação do algoritmo de Dijkstra na malha menos refinada coincide com a quantidade de trajetórias ótimas encontradas com a hipótese de trajetórias retilíneas. Observando estas trajetórias mais detalhadamente, conclui-se que de fato são as mesmas e que a aplicação de Dijkstra na malha mais pobre apenas encontrou as trajetórias ótimas quando estas são retilíneas. Dessa forma, em uma seção com maior área danificada, é possível que a aplicação de Dijkstra em uma malha mais pobre não gere resultados da mesma qualidade que os observados neste caso.

Com isso, a aplicação do próprio algoritmo de Dijkstra em malhas de nós mais pobres pode ser vista como uma alternativa no fornecimento de resultados aproximados. Essa alternativa ainda tem a vantagem de apresentar tempos de processamento muito menores, o que se deve à grande dependência do algoritmo de Dijkstra com relação ao número de nós da malha utilizado no mapeamento (essa dependência será explorada posteriormente).

Em resumo, no que se refere à qualidade das soluções, os algoritmos determinísticos visivelmente são superiores, já que sempre encontram as soluções ótimas para o problema das trajetórias. Enquanto isso, os algoritmos GA e ACS possuem natureza probabilística, trabalhando com uma tolerância na qualidade das soluções em prol de um tempo de processamento competitivo. Dessa forma, as respostas encontradas por estes métodos são aproximadas. Como alternativa à aplicação de algoritmos bioinspirados na obtenção de soluções aproximadas, existe a possibilidade de aplicar algoritmos determinísticos como Dijkstra em malhas de nós mais pobres.

Em seguida, avalia-se o tempo de processamento, segundo aspecto a ser levado em consideração na avaliação dos algoritmos. Após a solução do exemplo, elaborou-se o gráfico da Figura 4.6. Este relaciona os tempos de processamento dos algoritmos, em segundos (escala vertical), com o número de nós da malha do exemplo 1 (escala horizontal), englobando todos os níveis de refinamento da malha da nós, que possui 2601 nós no nível 10, o mais alto. Dessa forma, os tempos dos algoritmos Dijkstra v1 (implementação inicial, anterior ao trabalho), Dijkstra v2, A*, GA com tolerância de 5% de erro médio e ACS com tolerância de 0,5% de erro médio são apresentados.

Figura 4.6 – Gráfico comparativo dos tempos de processamento de todos os algoritmos (exemplo 1).



Fonte: Autor.

Apesar dos resultados em tempo absoluto serem diferentes de acordo com as especificações de cada máquina onde o *software* for executado, o formato das curvas e consequentemente a diferença percentual entre elas se mantém, visto que o código é sequencial (não paralelizado). É importante salientar que em todos os algoritmos, exceto em Dijkstra v1, inclui-se o tempo de pré-processamento necessário para a

montagem do grafo antes da determinação das trajetórias. Em Dijkstra v1, o processo de montagem do grafo é feito concomitantemente à aplicação do algoritmo.

Além disso, foram adicionados marcadores de tempo no *software*, de forma a poder comparar o tempo de execução de cada uma das etapas dos algoritmos: pré-processamento (quando existe), tempo de viagem obtido pelo algoritmo em cada trajetória (para comparar a qualidade das trajetórias encontradas em cada método) e tempo total de aplicação. Retirando estes marcadores, o desempenho dos algoritmos seria superior, mas como estes estão presentes em todos os códigos, não existe nenhum com desvantagem nesse aspecto.

Observando a Figura 4.6, nota-se que o tempo de processamento da implementação Dijkstra v1 cresce exponencialmente quando comparado aos outros algoritmos, atingindo tempo pouco superior a 20.000 segundos (cerca de 5 horas e 30 minutos) para o maior nível de refinamento de malha de nós, enquanto o maior tempo dentre os outros algoritmos é pouco maior que 2.000 segundos (cerca de 30 minutos). Este comportamento já era esperado, já que Dijkstra v1 é uma implementação inicial, anterior a este trabalho.

A grande diferença do tempo de execução da implementação inicial Dijkstra v1 em relação aos novos algoritmos implementados demonstra a qualidade das soluções propostas quando comparadas ao que era utilizado no *software* TUSom. O aumento acentuado do tempo de execução deve-se aos fatores já descritos na seção 3.1:

- Ausência de diretivas de otimização intrínsecas do compilador;
- Montagem do grafo no cálculo de cada trajetória, o que é desnecessário, dada a natureza estática do problema (o grafo é o mesmo para o cálculo de todas as trajetórias e não se altera ao longo do tempo, podendo ser montado uma única vez);
- Cálculo e montagem de toda a matriz que armazena os pesos das arestas do grafo (tempos de viagem entre nós da malha), que também são desnecessários, dada a simetria dessa matriz, já que o grafo não é dirigido (custo entre os nós i e j é igual ao custo entre j e i);
- Processos lentos de ordenação dos tempos de viagem entre o nó atual e todos os outros nós não visitados. Estes processos são repetidos a cada iteração do método;

- Execução do algoritmo de Dijkstra até que todos os nós da malha sejam visitados, o que também é desnecessário, visto que este pode ser interrompido quando se atinge o nó destino (a única grandeza de interesse em cada trajetória é o tempo de viagem entre o nó origem – transdutor emissor – e o nó destino – transdutor receptor).

Ignorando questões de implementação e analisando o algoritmo em si, baseando-se também nas definições da seção 2.6.1, é possível inferir que o algoritmo de Dijkstra apresenta grande dependência com relação ao tamanho da malha de nós adotada na seção, pois o problema das trajetórias é semelhante ao TSP. Dessa forma, o conjunto de dados do problema e consequentemente do algoritmo (devido a suas características) cresce exponencialmente, pois o número de possibilidades de visita é $(n+1)!$, sendo n o número total de nós da malha. O termo extra deve-se à presença do transdutor receptor, que é um nó acrescentado à malha inicial durante a montagem do grafo. O grande número de possibilidades existe porque cada nó visitado elimina apenas uma opção dentro de um espaço de buscas muito grande.

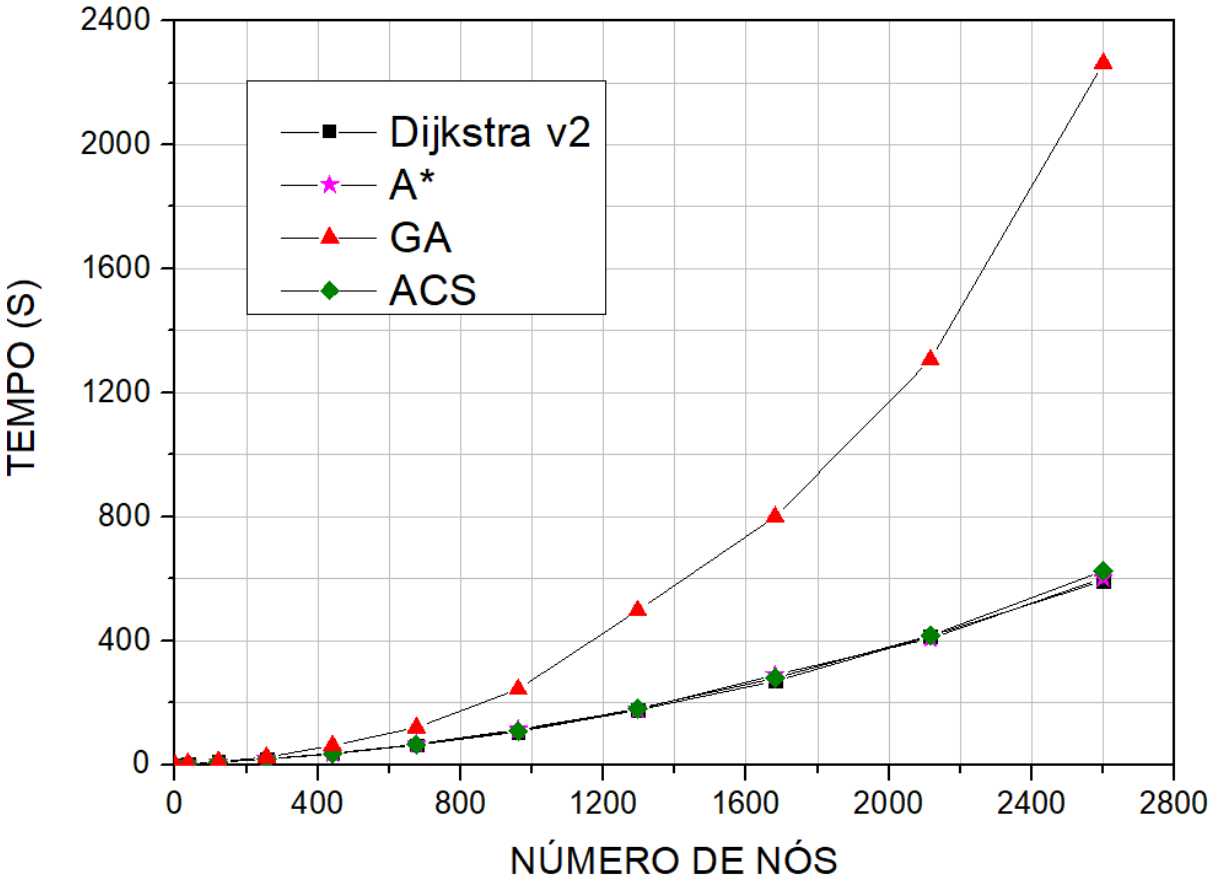
Como o alto tempo de processamento da implementação Dijkstra v1 não permite a visualização adequada dos tempos de execução dos outros algoritmos, este foi excluído e a escala vertical (tempo de processamento em segundos) foi reajustada, dando origem à Figura 4.7.

Observando a figura, novamente é possível enxergar um algoritmo que possui desempenho inferior aos outros: o GA. Nas 3 primeiras malhas de nós os algoritmos apresentam tempos de execução muito próximos (escala dificulta a visualização), mas na malha mais refinada, com 2601 nós, o GA chega a apresentar tempos de execução de pouco mais de 2200 segundos (cerca de 35 minutos), enquanto os outros algoritmos apresentaram tempos muito próximos, na faixa de 600 segundos (cerca de 10 minutos). Essa diferença é considerável, ainda que muito menor que a diferença entre estes algoritmos e a implementação inicial Dijkstra v1.

Analisando os possíveis motivos dessa diferença, existe a própria adequação do algoritmo ao problema que se deseja resolver. Enquanto o GA é um método mais generalista, que serve para resolver inúmeros problemas de otimização, os algoritmos de Dijkstra, A* e ACS partiram da ideia de minimizar caminhos e foram exaustivamente testados no TSP, problema ao qual se adequaram muito bem. E, como visto

anteriormente, o TSP possui muitos pontos em comum com o problema da determinação de trajetórias ótimas de pulsos ultrassônicos.

Figura 4.7 – Gráfico comparativo dos tempos de processamento dos algoritmos, exceto Dijkstra v1 (exemplo 1).



Fonte: Autor.

Outros motivos para a grande diferença podem ser escolhas ruins do modelo de indivíduo e dos operadores adotados para o GA, já que estes possuem muitas variações. Um exemplo é a adoção do operador de cruzamento uniforme, que se mostra eficiente no que diz respeito à qualidade dos resultados, mas custoso computacionalmente quando comparado a outros operadores mais simples, como são os cruzamentos de um ou dois pontos (LINDEN, 2006). A sua adaptação à morfologia do GA baseado em ordem o torna ainda mais custoso, já que os genes não podem ser simplesmente trocados entre pais para gerar o filho, devido à restrição existente no problema da não repetição de nós.

O operador de mutação utilizado (mistura de sublistas), apesar de fornecer bons resultados, também é mais devagar que outros operadores disponíveis, como a

mutação espontânea de um único gene. Além disso, o operador de mutação também deve levar em conta as restrições de não repetição de nós do problema. No entanto, a aplicação do operador de mutação é menos complexa que a do operador de cruzamento, provavelmente o maior responsável pelo tempo superior. A probabilidade de mutação inicialmente adotada também pode ser alta demais (10%), desordenando boas soluções geradas pelo operador de cruzamento ao invés de favorecer a busca global de soluções (*exploration*).

Também é possível que o modelo de criação de indivíduos adotado não seja o mais conveniente, já que cada indivíduo é um vetor com todos os nós elegíveis para visita, visando a facilidade de aplicação dos operadores (todos indivíduos possuem o mesmo tamanho) e o cumprimento da condição de não repetição de nós em um mesmo indivíduo. Isso faz com que, quando a malha de nós apresenta refinamento máximo, cada indivíduo possua 2601 genes, ou seja, é representado por um vetor com 2601 termos. Além disso, os genes que aparecem após o nó do transdutor destino não acrescentam na avaliação dos indivíduos, mas exigem memória para armazenamento, instruções para criação e são operados normalmente. Um modelo com indivíduos menores, por exemplo, pode requerer menos esforço computacional.

Dessa forma, a utilização do algoritmo genético não se justificou no exemplo 1, já que ele tolera soluções com até 5% de erro médio e ainda exige mais tempo de processamento que as outras alternativas propostas, exceto Dijkstra v1, superado com folga por todos os novos algoritmos implementados.

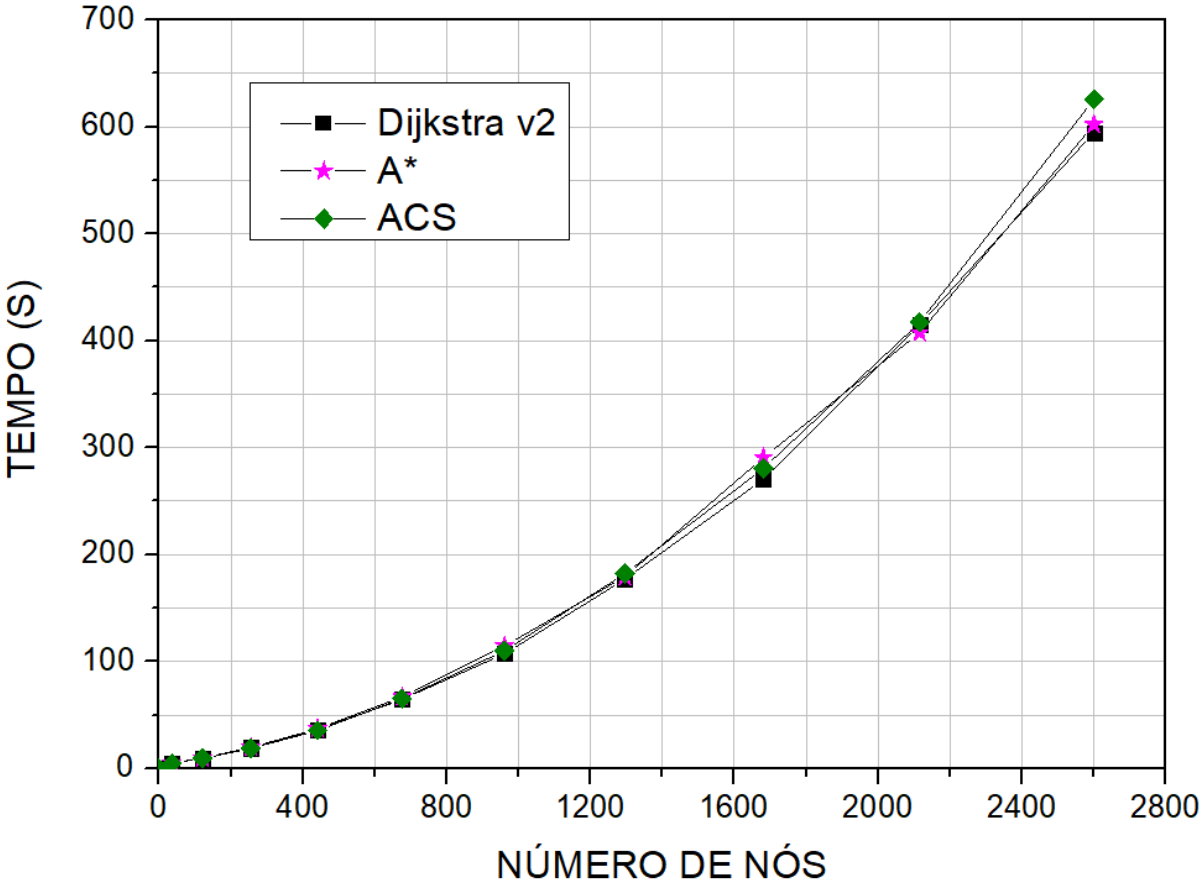
Assim como o algoritmo de Dijkstra, o GA também depende da malha de nós da seção, que influencia a criação e o tamanho de cada indivíduo no modelo adotado, além da quantidade de instruções que cada operador realizará em sua aplicação. No entanto, essa dependência é muito menor e o GA é mais influenciado pelo número de indivíduos e gerações adotados em sua execução (detalhados posteriormente).

Eliminando o GA do gráfico e reajustando a sua escala vertical, de forma análoga ao que foi feito anteriormente, temos como resultado a Figura 4.8. Novamente, o intuito é visualizar melhor a diferença de tempo entre os algoritmos restantes, já que a escala da figura anterior atrapalha esta tarefa.

Os algoritmos Dijkstra v2, A* e ACS despontam como as melhores opções na resolução do problema das trajetórias para o exemplo 1. Analisando a figura, é possível confirmar o que foi visto nas figuras anteriores, constatando que os algoritmos apresentam tempos de execução muito próximos e pequenos, entre 575 e

625 segundos (entre 9,5 e 10,5 minutos, aproximadamente) para a malha de nós com maior refinamento (2601 nós), o que destaca novamente a grande diferença entre as melhores e as piores alternativas na resolução do problema das trajetórias.

Figura 4.8 – Gráfico comparativo dos tempos de processamento dos algoritmos, exceto Dijkstra v1 e GA (exemplo 1).



Fonte: Autor.

Nos níveis 1 a 5 da malha nós (até cerca de 700 nós), os tempos dos algoritmos são muitos semelhantes e os pontos praticamente se sobrepõem (escala da figura dificulta a visualização). A partir disso, é possível enxergar diferenças entre eles: no nível 6, o algoritmo de Dijkstra é o mais rápido, acompanho de forma muito próxima pelo ACS e pelo A*, nesta ordem. No nível de refinamento 7, a posição de Dijkstra se mantém, enquanto A* e ACS invertem-se, assumindo a segunda e terceira posição, respectivamente. As diferenças continuam sendo muito pequenas entre eles. No nível de refinamento 8, o que foi visto no nível 6 se repete, com diferenças mais pronunciadas entre os algoritmos. No nível 9, A* aparenta ser o mais rápido, seguido por Dijkstra e ACS, mais uma vez com diferenças mínimas entre eles. Por fim, no nível

mais alto da malha de nós (10), Dijkstra é novamente mais rápido, seguido de perto por A*, enquanto a diferença para o ACS, mais uma vez o mais lento entre os remanescentes, parece aumentar.

Tendo como referência a interpretação visual do gráfico, o algoritmo de Dijkstra é mais rápido em um número maior de instâncias do problema, mais especificamente em 4 das 5 onde a diferenciação visual é possível. Com isso, ele constitui, no momento, a melhor abordagem para o problema das trajetórias. Apesar disso, A* apresenta tempos muito próximos e também se mostra uma ótima alternativa. Visando confirmar qual é o algoritmo mais rápido, análises posteriores serão realizadas usando tabelas. O algoritmo ACS demonstra o grande potencial das alternativas bioinspiradas.

O único ponto do algoritmo A* que pode ser alterado buscando obter desempenhos superiores é o cálculo da parcela de informação heurística da equação 2.20. No trabalho, adotou-se o tempo de viagem em linha reta entre o nó vizinho e o nó destino, considerando a maior velocidade presente na seção, como informação heurística (equação 3.2). Esse método foi adotado tendo em vista a simplicidade de cálculo e a consistência da heurística (sempre fornece a solução ótima).

Quando se trata do algoritmo de Dijkstra, este possui uma variação significativa de estrutura, chamada bidirecional. Em Dijkstra bidirecional, ao invés de partir da origem em direção ao destino, parte-se da origem em direção ao destino e do destino em direção à origem, alternando-os iteração após iteração ao longo do processo de busca. O resultado é que, ao invés de gerar um círculo com todos os nós visitados da origem até o destino (como na Figura 2.26), geram-se dois círculos menores e o algoritmo finaliza quando estes se encontram, já que nesse momento surge o caminho mínimo desejado. A alteração do seu princípio de funcionamento diminui o número de nós visitados, assim como ocorre em A*, com a vantagem de não haver o custo adicional do cálculo da informação heurística. A sua desvantagem consiste na menor simplicidade do algoritmo, que pode gerar um código menos enxuto e consequentemente menos eficiente.

Como o ACS não foi o algoritmo mais rápido e fornece soluções aproximadas (erro médio de até 0,5%), o seu uso também não se justificou na resolução do problema das trajetórias para o exemplo 1, dadas as estruturas de algoritmo implementadas e os parâmetros adotados. No entanto, o ACS se mostrou competitivo em tempo de processamento quando comparado aos algoritmos Dijkstra v2 e A*,

tendo grande potencial. Um ajuste melhor de seus parâmetros ou a tolerância de erros maiores podem fazer com que ele apresente desempenho superior aos demais algoritmos.

O resultado positivo do algoritmo ACS também abre precedentes para a aplicação futura de outros tipos de algoritmos de otimização por colônia de formigas, como o *MMAS*, o *Rank-based AS* ou o *Population-based AS*, já apresentados na seção 2.7.1. O *MMAS* tende a apresentar respostas de melhor qualidade quando comparado com o ACS, em troca de um tempo de convergência maior. Ambas são as variações de melhor desempenho e mais presentes em publicações, seguidas pelo *Rank-based AS* (DORIGO; STÜTZLE, 2004). Enquanto isso, o *Population-based AS* é uma variação mais recente que tem mostrado resultados competitivos na resolução do TSP (OLIVEIRA *et al*., 2017).

Compilando os resultados das Figuras 4.6, 4.7 e 4.8, elaborou-se a Tabela 4.3, tornando possível a análise os dados em maior detalhe. A tabela apresenta o tempo de execução de cada algoritmo (em segundos) em cada uma de suas colunas. Em suas linhas, variou-se o nível de refinamento da malha da nós. A melhor alternativa para cada malha (menor tempo de execução) é destacada na cor vermelha.

Tabela 4.3 – Tempo total dos algoritmos utilizados no exemplo 1, em função do tamanho da malha de nós da seção.

Refinamento da malha de nós	**Número de nós**	**TEMPO TOTAL DO ALGORITMO (seg.)**				
		Dijkstra v1	**Dijkstra v2**	**A***	**GA 5%**	**ACS 0,5%**
1	36	7,751	5,046	5,124	5,362	4,882
2	121	57,836	9,557	9,876	10,913	9,369
3	256	195,971	19,181	19,678	24,726	18,801
4	441	584,921	36,440	37,210	62,751	35,595
5	676	1364,438	64,656	66,974	120,763	65,267
6	961	2746,294	107,419	114,523	244,002	110,082
7	1296	4987,210	176,981	179,282	497,932	182,244
8	1681	8391,921	270,471	290,810	800,079	280,483
9	2116	13371,048	414,939	407,480	1305,873	417,081
10	2601	20182,649	594,593	602,191	2262,255	625,743

Fonte: Autor.

Partindo da tabela, nota-se que o algoritmo de Dijkstra foi superior aos demais em 5 instâncias do problema, nos níveis 5, 6, 7, 8 e 10 de refinamento da malha de nós. O algoritmo ACS foi o mais rápido nas 4 instâncias iniciais do problema (níveis 1 a 4). O algoritmo A* foi superior em apenas 1 instância do problema, no nível 9.

Analisando a tabela, é possível observar detalhes que escaparam à percepção nas figuras anteriores, provavelmente devido à dificuldade de se enxergar o resultado de tempo de processamento em malhas menores na Figura 4.8. O primeiro deles é que o algoritmo ACS foi o melhor dentre todas as alternativas nas malhas de nós 1 a 4. Esse fato reforça o potencial já atribuído ao algoritmo na análise das figuras. À partir do nível 4, seu desempenho cai, mas ele permanece sendo competitivo em relação aos algoritmos mais rápidos em todos os níveis de malha de nós.

A Tabela 4.3 também reforça hipóteses já levantadas na observação da Figura 4.8. Um exemplo é a superioridade de Dijkstra v2, o algoritmo que mais vezes apresentou o menor tempo de processamento (5). Além disso, os melhores desempenhos foram obtidos nas malhas de nós mais refinadas, que constituem situações de maior interesse prático, por representarem trajetórias do pulso mais próximas das reais.

Outra ideia já existente e reforçada pela tabela é o péssimo desempenho da implementação inicial, que resulta em tempos da ordem de grandeza de horas, enquanto os melhores algoritmos trabalham na ordem dos minutos. Por fim, vê-se também que a implementação do GA é consideravelmente inferior aos algoritmos Dijkstra v2, A* e ACS, mesmo aplicado com a maior tolerância de erro médio entre eles (5%). Apesar disso, essa implementação ainda é muito superior à Dijkstra v1, atingindo tempos cerca de 9 vezes menores na maior malha de nós.

O algoritmo A*, apesar de se mostrar competitivo com relação aos outros dois (Dijkstra v2 e ACS) ao longo de todos os níveis de refinamento da malha de nós, é superior em apenas uma instância do exemplo 1. No entanto, seu desempenho pode estar sendo ocultado por Dijkstra v2, já que a Tabela 4.3 exibe os tempos de todos algoritmos. Para determinar o segundo algoritmo com melhor desempenho, elaborou-se a Tabela 4.4. Nesta, compararam-se apenas os tempos de execução (para o exemplo 1) de A* e ACS, calculando a diferença percentual entre eles (quanto o primeiro é mais lento ou mais rápido que o segundo), informação presente na última coluna da tabela. Quando o primeiro é mais rápido, a diferença é destacada na cor

verde e utiliza-se o sinal negativo. Caso contrário, utiliza-se o sinal positivo e a informação é vermelha.

Tabela 4.4 – Comparação entre os tempos dos algoritmos A* e ACS no exemplo 1.

Refinamento da malha	Número de nós	Tempo A* (seg.)	Tempo ACS (seg.)	Diferença
1	36	5,124	4,882	**+ 4,96%**
2	121	9,876	9,369	**+ 5,41%**
3	256	19,678	18,801	**+ 4,66%**
4	441	37,210	35,595	**+ 4,54%**
5	676	66,974	65,267	**+ 2,62%**
6	961	114,523	110,082	**+ 4,03%**
7	1296	179,282	182,244	**- 1,63%**
8	1681	290,810	280,483	**+ 3,68%**
9	2116	407,480	417,081	**- 2,30%**
10	2601	602,191	625,743	**- 3,76%**

Fonte: Autor.

Analisando a tabela, nota-se que ACS é superior em 7 instâncias do exemplo, as 6 primeiras e o nível 8 da malha de nós. Enquanto isso, A* é mais rápido nos níveis 7, 9 e 10 da malha de nós. A maior diferença positiva entre ambos ocorre no nível 2, quando A* é 5,41% mais lento que ACS. A maior diferença negativa ocorre no nível 10, quando A* é 3,76% mais rápido que ACS.

À partir da tabela é possível concluir, num primeiro momento, que ACS é o segundo melhor algoritmo para a resolução do problema das trajetórias, já que é superior em 7 instâncias, contra 3 de A*. Além disso, quando A* ele é mais lento, as diferenças são maiores do que quando ele é mais rápido. No entanto, a superioridade de ACS ocorre predominantemente nos níveis de refinamento mais baixos, o que pode indicar um desempenho superior de A* em malhas maiores. Essa hipótese é reforçada pelas maiores diferenças percentuais, já que a maior superioridade de ACS é no nível 2, enquanto a maior superioridade de A* é no nível 10.

Quando comparamos a qualidade das respostas, A* é sempre superior, devido a sua natureza determinística. No entanto, como o erro associado ao algoritmo ACS é praticamente desprezível (0,5%), este algoritmo será mantido na segunda posição, pelo menos por enquanto.

Dessa forma, após a análise das Figuras 4.6, 4.7 e 4.8 e das Tabelas 4.3 e 4.4, resultados obtidos no exemplo 1, a classificação dos algoritmos de acordo com os critérios de qualidade de soluções obtidas e tempo de processamento é a seguinte: Dijkstra v2, ACS, A*, GA e Dijkstra v1. Os três primeiros algoritmos competem de forma muito próxima nos dois aspectos, seguidos do GA, que é consideravelmente mais lento e fornece as soluções com maior erro médio. No fim da lista, encontra-se Dijkstra v1, implementação anterior a este trabalho, que, apesar de fornecer soluções ótimas, é muito inferior às outras opções em tempo de processamento.

Com o objetivo de comparar os algoritmos aos pares com a melhor alternativa proposta até então (Dijsktra v2) e visualizar melhor as diferenças de tempos entre eles, novas tabelas foram elaboradas. A primeira delas é a Tabela 4.5, que compara A* e Dijkstra v2, de maneira análoga à Tabela 4.4. Como ambos algoritmos são determinísticos e encontram as soluções ótimas, são equivalentes no quesito qualidade das soluções obtidas.

Tabela 4.5 – Comparação entre os tempos dos algoritmos A* e Dijkstra v2 no exemplo 1.

Refinamento da malha	Número de nós	Tempo A* (seg.)	Tempo Dijkstra v2 (seg.)	Diferença
1	36	5,124	5,046	**+ 1,55%**
2	121	9,876	9,557	**+ 3,34%**
3	256	19,678	19,181	**+ 2,59%**
4	441	37,21	36,44	**+ 2,11%**
5	676	66,974	64,656	**+ 3,59%**
6	961	114,523	107,419	**+ 6,61%**
7	1296	179,282	176,981	**+ 1,30%**
8	1681	290,81	270,471	**+ 7,52%**
9	2116	407,48	414,939	**- 1,80%**
10	2601	602,191	594,593	**+ 1,28%**

Fonte: Autor.

Analisando a tabela, a superioridade de Dijkstra v2 fica evidente, já que este é mais rápido que A* em 9 das 10 instâncias do exemplo 1, tanto nas maiores quanto nas menores malhas. A* é superior apenas no nível 9. A maior diferença positiva entre ambos ocorre no nível 8, quando A* é 7,41% mais lento que ACS. A maior diferença negativa ocorre no nível 9, quando A* é 1,80% mais rápido que ACS. Não é possível

observar tendências de aumento ou diminuição na diferença entre os algoritmos conforme a malha de nós aumenta. No entanto, devido à superioridade pronunciada de Dijkstra v2, é provável que ele também possua desempenhos melhores em malhas com mais nós que o exemplo 1.

Como o algoritmo Dijkstra v2 foi mais rápido que A* na maioria das instâncias, é possível concluir que o maior número de visitas nodais do primeiro algoritmo compensa o custo computacional de cálculo da heurística do segundo. Nesse caso, a implementação de Dijkstra bidirecional provavelmente retornaria desempenhos ainda melhores.

Visando comparar Dijkstra v2 com o algoritmo ACS, que demonstrou grande potencial em sua aplicação inicial, elaborou-se a Tabela 4.6 de forma análoga às Tabelas 4.4 e 4.5. Agora, também são apresentados os parâmetros variáveis do algoritmo ACS: número de vizinhos adotados na *candidate list* (CL) e número de iterações. Os valores dos outros parâmetros já foram definidos na seção 3.4 e são fixos ($m = 10$, $\beta = 2$, $\xi = \rho = 0{,}1$, $q_0 = 0{,}5$ e $\tau_0 = 1/n.C^{nn}$).

Também mostrou-se o erro médio obtido em cada nível de refinamento da malha de nós. É importante destacar que este erro é obtido comparando-se os tempos de viagem dos pulsos ultrassônicos entre o ACS e o algoritmo Dijkstra v2 para um mesmo nível da malha de nós, e não tomando como referência a malha mais refinada. Isso é feito visando comparar a aplicação dos algoritmos nas mesmas condições.

Tabela 4.6 - Comparação entre os tempos dos algoritmos ACS e Dijkstra v2 no exemplo 1.

Refinamento da malha	Número de nós	Nº de vizinhos	Nº de iterações	Erro médio	Tempo ACS (seg.)	Tempo Dijkstra v2 (seg.)	Diferença
1	36	2	5	0,34%	4,882	5,046	**- 3,25%**
2	121	2	5	0,26%	9,369	9,557	**- 1,97%**
3	256	2	5	0,41%	18,801	19,181	**- 1,98%**
4	441	2	5	0,36%	35,595	36,440	**- 2,32%**
5	676	2	5	0,30%	65,267	64,656	**+ 0,95%**
6	961	2	5	0,32%	110,082	107,419	**+ 2,48%**
7	1296	2	5	0,31%	182,244	176,981	**+ 2,97%**
8	1681	2	5	0,35%	280,483	270,471	**+ 3,70%**
9	2116	2	5	0,36%	417,081	414,939	**+ 0,52%**
10	2601	2	5	0,35%	625,743	594,593	**+ 5,24%**

Fonte: Autor.

Quando se trata de qualidade das soluções, Dijkstra v2 é sempre superior ao ACS, visto que retorna as soluções ótimas em todas execuções. No que diz respeito ao tempo de processamento de cada um dos algoritmos (em segundos), é possível observar na tabela que o ACS é superior ao algoritmo Dijkstra v2 nas 4 instâncias iniciais do exemplo 1, como já visto anteriormente na Tabela 4.3, indicando uma superioridade em malhas menores. Nas outras 6 instâncias (nível 5 de malha de nós em diante), o algoritmo Dijkstra v2 é melhor, indicando superioridade em malhas maiores. Como Dijkstra v2 é mais rápido em um número maior de níveis e estes níveis são de maior interesse (trajetórias mais próximas das reais), ele também é considerado superior no aspecto tempo de processamento.

A maior diferença de tempo positiva entre os algoritmos ocorre na malha de refinamento 10 (2601 nós), quando ACS é 5,24% mais lento que Dijkstra v2. A maior diferença de tempo negativa ocorre na malha de refinamento 1 (36 nós), quando ACS é 3,25% mais rápido que Dijkstra v2. Essas diferenças reforçam a ideia de que ACS é superior em pequenas malhas e Dijkstra v2 em grandes malhas de nós. Assim como na Tabela 4.5, não é possível enxergar uma tendência de aumento ou diminuição das diferenças de tempo entre os algoritmos conforme a malha de nós aumenta. No entanto, é clara a tendência de crescimento do tempo em maior proporção para o ACS, já que este inicia a aplicação do algoritmo mais rápido, essa diferença diminui e a tendência se inverte conforme cresce o nível de refinamento da malha de nós, ou seja, ACS é cada vez mais lento que Dijkstra v2.

O ACS depende do tamanho da malha de nós em etapas como a montagem do grafo, mas depende em maior proporção dos parâmetros variáveis do algoritmo. Agora, os parâmetros mais importantes são o número de formigas, de iterações e o tamanho da CL, que foram relativamente pequenos: apenas 10 formigas (valor fixado), 2 vizinhos e 5 iterações foram suficientes para encontrar trajetórias com erro médio de 0,35% na malha de nós mais refinada (2601 nós). O fato do número mínimo de vizinhos na CL e de iterações ser suficiente para resolver todos os níveis de malha pode indicar dois fatos: um exemplo simples demais ou ótimas adequação da estrutura do ACS ao problema das trajetórias, calibração dos parâmetros e definição da informação heurística, já que o algoritmo gerou soluções iniciais de grande qualidade.

Ainda na malha de nós com maior refinamento, quando analisamos a qualidade das trajetórias obtidas pelo algoritmo ACS, de acordo com a Tabela 4.7, temos que o

maior erro obtido dentre todas as trajetórias do exemplo 1 foi de 4,08%, um erro baixo. Além disso, o erro mínimo obtido, de 0,00%, representa os casos em que as trajetórias ótimas foram encontradas. Agora, o erro nulo foi encontrado em 74 trajetórias, em comparação com 60 para a hipótese de trajetórias retilíneas. Portanto, tem-se um número maior e que representa 49,33% do total de trajetórias (150), indicando que o ACS foi capaz de encontrar trajetórias ótimas mesmo quando estas não eram retilíneas.

Tabela 4.7 – Erros no exemplo 1 considerando mapeamento com o algoritmo ACS na malha mais refinada (2601 nós).

Erros no exemplo 1 - Algoritmo ACS	
Número de trajetórias	150
Número de nós da malha	2601 (máximo)
Erro médio	0,35%
Erro máximo	4,08%
Erro mínimo	0,00%
Nº de trajetórias com erro mínimo	74 (49,33% do total)
Desvio padrão	0,79%

Fonte: Autor.

Por fim, ainda no nível 10 da malha de nós, o desvio padrão das medidas de tempo foi de 0,79%, valor 125% maior que o erro médio de 0,35% obtido na malha, indicando um conjunto de dados que varia bem com relação à média, mas com valores de erro absoluto pequenos. Dessa forma, conclui-se que o algoritmo ACS encontra respostas com ótima qualidade, ou seja, pontos subótimos com erros baixos na maioria das trajetórias. Isso é realizado em tempos bastante satisfatórios, que competem na mesma ordem de grandeza com os melhores algoritmos. Assim, a sua aplicação inicial foi bastante promissora.

Visando confirmar as observações feitas à partir da Tabela 4.3 e da Figura 4.7 e enxergar a variação dos parâmetros do GA, elaborou-se a Tabela 4.8, de maneira análoga à anterior, comparando o algoritmo Dijkstra v2 e o GA. Agora, os parâmetros exibidos são o número de indivíduos e de gerações adotados para o GA em cada nível da malha de nós, com tolerância de erro médio de até 5%. Os outros parâmetros do algoritmo foram definidos na seção 3.3 (PC = 0,9, PA = 0,1 e 2 elites por geração). Assim como na tabela anterior, o erro médio é obtido comparando-se os tempos de

viagem dos pulsos ultrassônicos entre o GA e o algoritmo Dijkstra v2 para um mesmo nível de malha de nós.

No que diz respeito ao tempo de processamento dos algoritmos, já era claro o desempenho significativamente inferior do GA. No entanto, a dimensão das diferenças não era tão clara. O GA é inferior em todas as instâncias do problema, se distanciando cada vez mais de Dijkstra v2. Essa tendência comprova que o ritmo de crescimento de instruções do GA é muito maior, sendo 6,26% mais lento que Dijkstra logo no nível 1 da malha de nós. Na malha mais refinada, o GA já apresenta tempo maior que o triplo de Dijkstra v2, sendo 280,47% mais lento que este.

Tabela 4.8 - Comparação entre os tempos dos algoritmos GA e Dijkstra v2 no exemplo 1.

Refinamento da malha	Número de nós	Indivíduos	Gerações	Erro médio	Tempo GA (seg.)	Tempo Dijkstra v2 (seg.)	Diferença
1	36	80	10	4,20%	5,362	5,046	**+ 6,26%**
2	121	50	10	4,70%	10,913	9,557	**+ 14,19%**
3	256	50	15	3,75%	24,726	19,181	**+ 28,91%**
4	441	90	15	4,77%	62,751	36,440	**+ 72,20%**
5	676	90	15	4,97%	120,763	64,656	**+ 86,78%**
6	961	110	20	3,01%	244,002	107,419	**+ 127,15%**
7	1296	110	20	2,75%	497,932	176,981	**+ 181,35%**
8	1681	110	20	3,07%	800,079	270,471	**+ 195,81%**
9	2116	120	20	3,27%	1305,873	414,939	**+ 214,71%**
10	2601	120	25	2,73%	2262,255	594,593	**+ 280,47%**

Fonte: Autor.

Assim como o ACS, o GA também depende em maior proporção dos seus parâmetros variáveis que do tamanho da malha de nós. Na Tabela 4.8, o número de indivíduos foi de 50 a 120. Enquanto isso, o número de gerações evoluiu de 10 para 25. Ou seja, o número de indivíduos aumentou 140% e o de gerações aumentou 150%, praticamente o mesmo crescimento para ambos.

Por outro lado, o número de nós da malha aumentou cerca de 71 vezes (de 36 para 2601 nós). Com isso, é possível concluir que o ritmo de crescimento dos parâmetros é muito menor que o da malha, um dado positivo. Este pequeno número de indivíduos e gerações também se deve ao erro tolerado, de 5% para o GA. Quando comparam-se os parâmetros variáveis do ACS e do GA, é possível visualizar que com menos indivíduos (10), vizinhos na CL (apenas 2) e iterações do método (apenas 5),

o ACS gera resultados muito superiores, com erro médio tolerável 10 vezes menor (0,5%, enquanto o do GA é de 5%). Este fato também influencia o desempenho computacional dos algoritmos e é uma das razões pelas quais o ACS é superior também em tempo de processamento.

No geral, o aumento do número de indivíduos e de gerações do GA geram diminuições semelhantes no erro médio, mas o número de indivíduos impacta menos no tempo de processamento. Dessa forma, ele foi preferido, quando possível.

Tratando da qualidade das trajetórias obtidas, de acordo com a Tabela 4.9, na malha de nós com maior refinamento o erro médio do GA foi de 2,73%. Apesar deste valor ser baixo, não é representativo da amostra, já que algumas trajetórias se desviam muito da ótima, chegando a apresentar erros de mais de 100%. A trajetória com o maior erro encontrado foi de 102,17%. No entanto, este erro extremo foi um caso pontual, já que a segunda trajetória de maior erro é 34,16% mais lenta que a ótima. Ainda assim, o valor do maior erro é consideravelmente superior até ao erro máximo obtido na hipótese de trajetórias retilíneas (31,42%).

Tabela 4.9 – Erros no exemplo 1 considerando mapeamento com o GA na malha mais refinada (2601 nós).

Erros no exemplo 1 - GA	
Número de trajetórias	150
Número de nós da malha	2601 (máximo)
Erro médio	2,73%
Erro máximo	102,17%
Erro mínimo	0,00%
Nº de trajetórias com erro mínimo	7 (4,67% do total)
Desvio padrão	9,18%

Fonte: Autor.

Desta forma, quando se utilizou o mapeamento com o GA, o erro médio das trajetórias foi menor que o das trajetórias retilíneas, mas o erro máximo foi notavelmente superior. Conclui-se então que o erro médio do GA é composto por muitas trajetórias com erros baixos e acaba sendo elevado por erros pontuais muito grandes. Uma alternativa para melhorar a qualidade dos resultados do algoritmo seria comparar, ao fim de sua aplicação, a melhor trajetória encontrada com a trajetória

retilínea. Caso a última seja mais rápida, por mais que atravesse trechos de heterogeneidade, é adotada. Assim, erros extremos podem ser evitados.

Novamente, a trajetória de erro mínimo encontrada foi de 0,00%. No entanto, esse valor de erro foi obtido apenas em 7 trajetórias (cerca de 5% do total), demonstrando que são muito raros os casos em que se encontram as trajetórias ótimas com os parâmetros utilizados. O desvio padrão das medidas foi de 9,18%, o que indica um conjunto de dados menos uniforme que o obtido no ACS (enquanto neste o desvio padrão era 125% maior que o erro médio obtido, no GA ele é 235% maior). O desvio maior é influenciado pelas medidas pontuais que destoaram muito da média, mencionadas anteriormente.

Em suma, é possível concluir que o uso do GA não se justificou na resolução do problema das trajetórias para o exemplo 1, já que apresentou resultados bastante inferiores tanto em tempo de processamento quanto em qualidade das soluções, com erros médios de até 5% e erro máximo maior que 100% quando comparadas às trajetórias ótimas. Dentre os quatro algoritmos implementados, o GA foi o que apresentou o pior desempenho, apenas à frente da implementação inicial Dijkstra v1, anterior ao trabalho. Apesar disso, o GA é muito superior a esta alternativa e outras estruturas do algoritmo podem ser testadas no futuro.

Por fim, a implementação inicial Dijkstra v1 também foi comparada com o algoritmo Dijkstra v2, de forma a ter ideia da ordem de grandeza da diferença dos tempos de processamento. Este resultado foi reunido na Tabela 4.10. Como ambos algoritmos são determinísticos, fornecem os menores tempos possíveis para a malha de nós construída (tempos de referência) e não existe erro envolvido.

A diferença exorbitante entre os algoritmos implementados e Dijkstra v1, já observada na Figura 4.6, fica comprovada pela tabela. No primeiro nível de refinamento de malha de nós, Dijkstra v1 é 53,61% mais lento que Dijkstra v2. A diferença de tempos cresce exponencialmente conforme a malha aumenta: no nível 10 da malha de nós, o tempo de Dijkstra v1 representa cerca de 34 vezes o tempo de Dijkstra v2, sendo 3294,36% mais lento.

A Tabela 4.8 também reforça a ideia de que todas as alternativas implementadas, ainda que bastante diferentes entre si em todos os aspectos (princípios de funcionamento, qualidade das respostas e tempo de processamento), são muito superiores ao processo de determinação das trajetórias adotado anteriormente no *software* TUSom (Dijkstra v1).

Tabela 4.10 – Comparação entre os tempos dos algoritmos Dijkstra v1 e v2 no exemplo 1.

Refinamento da malha	Número de nós	Tempo Dijkstra v1 (seg.)	Tempo Dijkstra v2 (seg.)	Diferença
1	36	7,751	5,046	**+ 53,61%**
2	121	57,836	9,557	**+ 505,17%**
3	256	195,971	19,181	**+ 921,69%**
4	441	584,921	36,440	**+ 1505,16%**
5	676	1364,438	64,656	**+ 2010,30%**
6	961	2746,294	107,419	**+ 2456,62%**
7	1296	4987,210	176,981	**+ 2717,94%**
8	1681	8391,921	270,471	**+ 3002,71%**
9	2116	13371,048	414,939	**+ 3122,41%**
10	2601	20182,649	594,593	**+ 3294,36%**

Fonte: Autor.

Finalizada a etapa de comparação dos tempos de processamento dos algoritmos, parte-se para outra questão. É conhecido que os algoritmos Dijkstra v2, A*, GA e ACS possuem etapas de pré-processamento, em que monta-se o grafo que representa a malha da seção transversal de concreto para a qual deseja-se mapear os pulsos. Tomando como referência a melhor alternativa na resolução do problema das trajetórias até o momento, o algoritmo Dijkstra v2, montou-se a Tabela 4.11, comparando o tempo de pré-processamento necessário com o tempo de aplicação do algoritmo, ambos em segundos.

Como a montagem do grafo é um processo muito semelhante para todos os algoritmos, considera-se que não seja necessária a criação de tabelas análogas para os outros e que esta é suficientemente representativa.

Observa-se na tabela que, enquanto na malha de menor refinamento do exemplo 1 (36 nós) o pré-processamento representa apenas 1,80% do tempo total do algoritmo, sendo praticamente 50 vezes menor que o tempo de aplicação, na malha de maior refinamento (2601 nós), o tempo de pré-processamento representa 80,13% do tempo total do algoritmo, sendo 4 vezes maior que o tempo de aplicação.

À partir disso, é direta a conclusão de que o tempo necessário à etapa de pré-processamento do algoritmo cresce em um ritmo muito maior que o da aplicação do algoritmo em si. Na transição do refinamento 5 para o refinamento 6 da malha, quando esta possuiria cerca de 725 nós, o tempo de pré-processamento passa a ser maior que o de aplicação do algoritmo.

Tabela 4.11 – Comparação dos tempos de pré-processamento e de aplicação do algoritmo Dijkstra v2 no exemplo 1.

Refinamento da malha	Número de nós	Tempo pré-process. (seg.)	Tempo algoritmo (seg.)	Tempo total (seg.)	Parcela do todo (pré-process.)
1	36	0,091	4,955	5,046	1,80%
2	121	1,011	8,546	9,557	10,58%
3	256	4,548	14,633	19,181	23,71%
4	441	13,500	22,940	36,440	37,05%
5	676	31,357	33,299	64,656	48,50%
6	961	61,493	45,926	107,419	57,25%
7	1296	115,641	61,340	176,981	65,34%
8	1681	189,470	81,001	270,471	70,05%
9	2116	317,285	97,654	414,939	76,47%
10	2601	476,460	118,133	594,593	80,13%

Fonte: Autor.

As grandes parcelas do tempo utilizadas na etapa de pré-processamento podem indicar que, para que se consigam diminuições posteriores no tempo total dos algoritmos estudados neste trabalho, um foco maior deve ser dado à esta etapa do código. Como o grafo que representa o problema das trajetórias é denso, optou-se pela sua representação na forma de uma matriz de adjacência, já que esta, apesar de ocupar mais memória, apresenta maior facilidade de indexação e chamada dos valores de tempo de viagem entre nós. Uma das alternativas para diminuir o tempo de execução da etapa de montagem do grafo seria a sua representação em outro formato, como o de listas de adjacência, já apresentadas na seção 2.5.2.

Outra alternativa para diminuir o tempo de execução dessa etapa seria adotar formas diferentes de criação de malha de nós para a seção transversal de concreto, adaptando a malha ao campo de velocidades específico que se deseja estudar. Dessa forma, uma malha mais pobre seria adotada em áreas onde existe um campo de velocidades mais uniforme, enquanto uma mais refinada seria adotada em áreas onde existe um gradiente maior de velocidades ou onde identifica-se que será um caminho de deslocamento preferencial dos pulsos ultrassônicos. Essa abordagem é semelhante ao que ocorre em modelos de elementos finitos, onde a malha de elementos é enriquecida em pontos de maior interesse como descontinuidades e/ou concentrações de tensões.

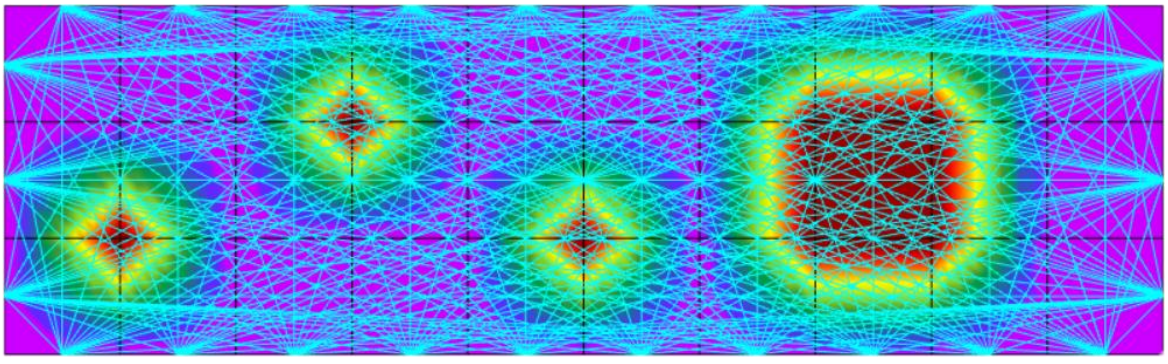
Novos modelos de criação de malhas podem ser muito vantajosos, já que, se bem executados, podem diminuir muito o seu número total de nós, eliminando os que forem julgados desnecessários em cada caso. Isso faz com que o grafo que representa o problema fique menor, o que diminui tanto o tempo de pré-processamento quanto o de aplicação dos algoritmos.

Para o exemplo 2, o procedimento de análise adotado será semelhante: inicialmente, serão feitas comparações visuais dos resultados do mapeamento das trajetórias. Em seguida, os dados de tempo de processamento serão apresentados na forma de gráficos e esmiuçados na forma de tabelas, comparando todas as alternativas. Determinada a melhor alternativa para a resolução do problema das trajetórias nos exemplos, esta será comparada aos pares com outros algoritmos. Por fim, o tempo de pré-processamento dessa alternativa será analisado. Como diferenças, agora serão feitas comparações constantes com o que foi obtido no exemplo 1 e extrapolações de conclusões, quando for possível fazê-las.

4.2 EXEMPLO 2

Realizando novamente a sobreposição das imagens do campo de velocidades e das trajetórias retilíneas, agora da Figura 3.6 (exemplo 2), temos a Figura 4.9. Mais uma vez, os pulsos ultrassônicos atravessam regiões de baixa velocidade do material, onde os defeitos se encontram, o que constitui um comportamento distante do real e que demonstra a importância da adoção de estratégias de mapeamento do pulso ultrassônico.

Figura 4.9 – Sobreposição do campo de velocidades e das trajetórias retilíneas no exemplo 2 (seção retangular).



Fonte: Autor.

Os dados da seção foram compilados na Tabela 4.12. As velocidades máxima e mínima não se alteraram em relação ao exemplo 1, sendo de 4500m/s e 2000m/s, respectivamente. Dessa forma, a diferença entre elas continua sendo a mesma e a velocidade mais baixa presente na seção é cerca de 55,56% menor que a mais alta.

Tabela 4.12 – Informações sobre o exemplo 2 considerando trajetórias retilíneas.

Informações do exemplo 2 - Trajetórias retilíneas	
Número de trajetórias	229
Velocidade máxima na seção	4500m/s
Velocidade mínima na seção	2000m/s
Área da seção com defeitos	58%
Área da seção íntegra	42%
Erro médio	14,90%
Erro máximo	34,55%
Erro mínimo	0,00%
Nº de trajetórias com erro mínimo	10 (4,37% do total)
Desvio padrão	9,90%

Fonte: Autor.

Quando comparamos o tempo de propagação dos pulsos ultrassônicos na seção utilizando a hipótese de propagação retilínea e os algoritmos determinísticos (na malha de nós com o maior nível de refinamento), o erro médio obtido é de 14,90% considerando as 229 trajetórias do exemplo, um valor considerável. Apesar disso, este erro é menor que o esperado, visto que cerca de 58% da área da seção apresenta algum tipo de dano. Enquanto a área danificada aumentou de 33% para 58% do exemplo 1 para o exemplo 2 (aumento de 25%), o erro médio supondo trajetórias retilíneas não seguiu o mesmo ritmo de crescimento, subindo de 9,31% para 14,90% (aumento de 5,59%). O grande número de trajetórias retilíneas ótimas no exemplo 1 também justifica o menor valor de erro médio.

O erro máximo obtido na seção foi de 34,55%, valor alto e maior que o dobro do erro médio. Quando comparado ao exemplo 1, houve um aumento ligeiro, já que antes o valor era de 31,42%. Além disso, no exemplo 1 o erro máximo era maior que o triplo da média, o que provavelmente indica um conjunto de dados menos uniforme. Isso se comprova através da comparação do desvio padrão, de 11,45% no exemplo 1 (maior que a média) e de 9,90% no exemplo 2 (5% menor que a média). Dessa

forma, é possível concluir que, apesar dos erros gerados por trajetórias retilíneas serem maiores no exemplo 2, o que era esperado devido à maior área danificada da seção, estes valores de erro variaram menos em torno da média.

O erro mínimo obtido na comparação foi de 0,00%, indicando que, em algumas trajetórias, a aproximação retilínea coincide com a melhor trajetória encontrada pelos algoritmos determinísticos. Do total de 229 trajetórias, este erro foi encontrado em apenas 10 (4,37% do total), o que indica que, neste exemplo, trajetórias retilíneas raramente coincidem com as ótimas, comportamento mais próximo do real e esperado numa seção mais danificada.

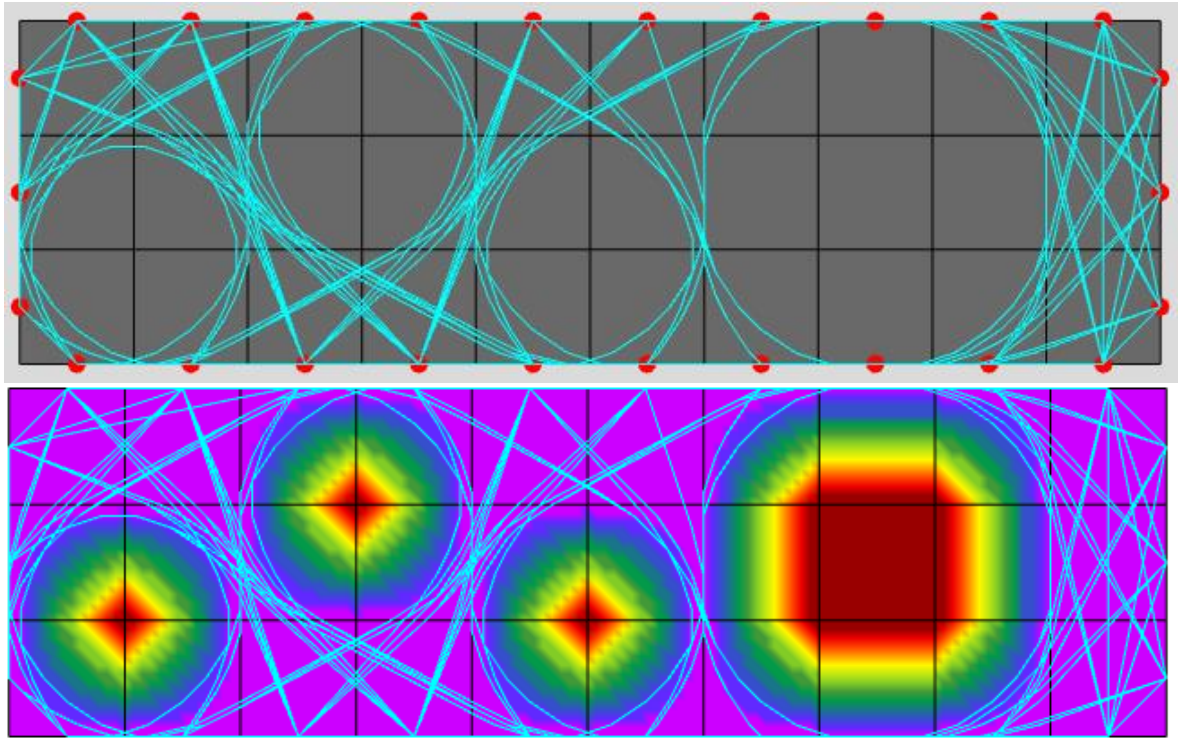
Após o mapeamento das trajetórias utilizando cada um dos algoritmos no exemplo 2 (seção retangular), para a malha de nós mais refinada, o resultado visual é apresentado nas Figuras 4.10, 4.11 e 4.12. É importante salientar que, tomando como limite superior o erro médio de 14,90% obtido nas trajetórias retilíneas (vide Tabela 4.12) trabalhou-se com uma tolerância de erro de 10% no GA e de 1% no ACS. A definição destes valores seguiu a mesma lógica do exemplo 1: os erros do GA foram fixados em aproximadamente 5% abaixo do gerado pelas trajetórias retilíneas, enquanto o ACS gerou soluções iniciais de muita qualidade, atingindo valores de erro muito baixos com poucas iterações e número de vizinhos (parâmetros variáveis do algoritmo). Os novos valores de erro representam o dobro dos adotados no exemplo 1, o que era esperado, já que a seção retangular é mais complexa: possui maior número de nós, mais defeitos e não é simétrica.

A Figura 4.10 apresenta o resultado do mapeamento para os algoritmos determinísticos (Dijkstra e A*). Novamente, o mapeamento foi bem-sucedido, mesmo considerando que esta seção possui mais defeitos e de tamanhos diferentes. É muito clara a tendência dos pulsos desviarem dos defeitos simulados, buscando áreas homogêneas de material para de deslocarem.

É visível mais uma vez e em quantidade muito maior a concentração de trajetórias em pequenos caminhos preferenciais, entre os danos, que conservaram o concreto íntegro. Algumas inclusive caminham pelas bordas da seção, devido à proximidade entre os defeitos e essas regiões, comportamento que não foi visto no exemplo 1 e pouco esperando na realidade, já que o concreto é mais irregular nas bordas, como consequência de contato com o ambiente externo e do seu processo de desforma. Por mais bem vibrado que o concreto tenha sido, sempre existirão vazios no material, mais evidentes nas regiões das bordas da seção.

Como a área danificada da seção é consideravelmente maior que no exemplo 1 (antes era de 33% e agora é de 58%), muitas das trajetórias coincidem em trechos consideráveis, buscando desviar dos danos. Em alguns casos, estas chegam a se confundir e, na totalidade da imagem, aparenta-se ter menos trajetórias que na Figura 3.6. Nas parcelas do material em que não houve dano, é possível observar que as trajetórias permaneceram praticamente retilíneas, apesar de serem em menor quantidade devido à maior área danificada. Ainda é possível observar que, mesmo para os algoritmos determinísticos (fornecem soluções ótimas), foram encontradas trajetórias que atravessam regiões de leve heterogeneidade (em azul na figura).

Figura 4.10 – Resultado do mapeamento do pulso ultrassônico no exemplo 2 (seção retangular) para os algoritmos de Dijkstra e A* (determinísticos).



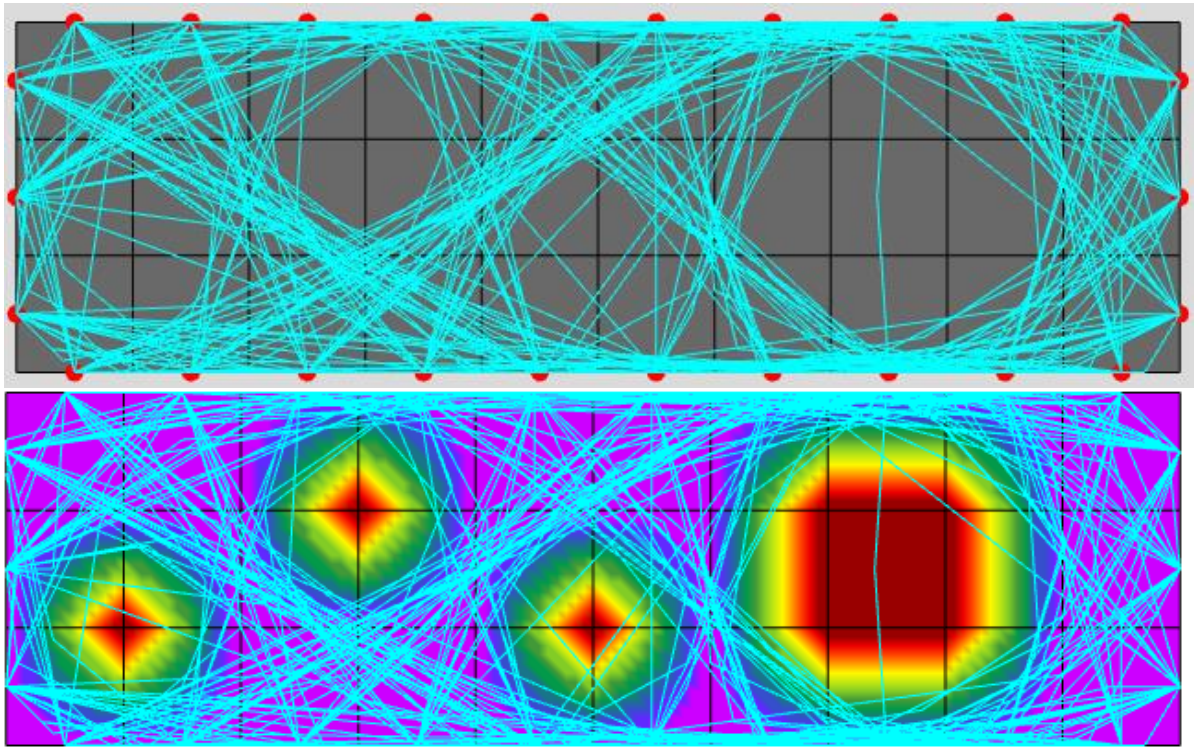
Fonte: Autor.

Na Figura 4.11, exibe-se o resultado obtido após a aplicação do GA. Comparando-a com a Figura 4.10, é visível a presença de menos trajetórias coincidentes e a seção aparenta ter mais trajetórias ao todo.

Além disso, agora a tolerância de erro atrelada ao algoritmo é maior, de 10% e é natural que menos trajetórias se sobreponham. Inclusive, é possível enxergar trajetórias ruins na figura, como a que atravessa o defeito grande e algumas outras que passam por regiões de heterogeneidade considerável (áreas amarelas e

vermelhas da figura). Esse comportamento não foi observado no exemplo 1, provavelmente devido à menor tolerância de erro dos algoritmos e à maior simplicidade do exemplo, que possuía menor área danificada. No exemplo 1, as trajetórias atravessaram apenas as regiões azul-escuras e verdes, que apresentam heterogeneidades leves.

Figura 4.11 - Resultado do mapeamento do pulso ultrassônico no exemplo 2 (seção retangular) para o algoritmo genético (GA).

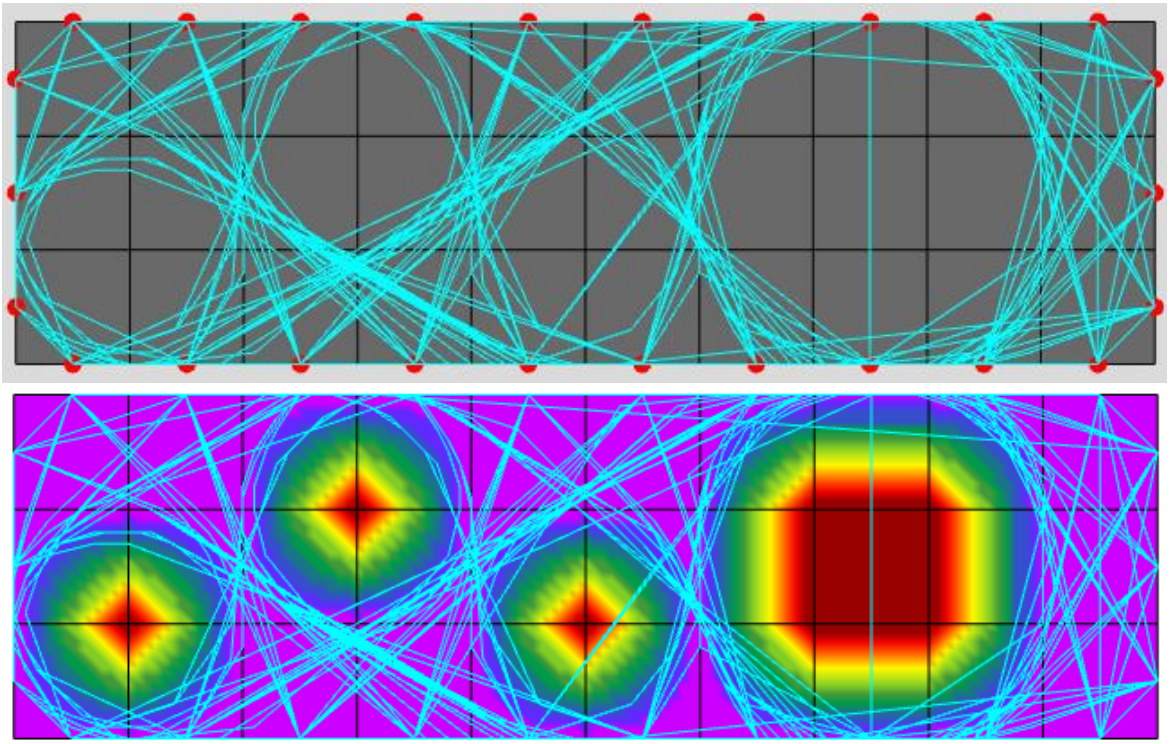


Fonte: Autor.

A Figura 4.12 apresenta o resultado do mapeamento após a aplicação do ACS. Quantitativamente, os resultados são consideravelmente diferentes com relação às trajetórias da Figura 4.11, visto que o GA opera com uma tolerância de erro de 10%, enquanto a tolerância de erro do ACS é de apenas 1%. No entanto, os resultados das figuras são semelhantes.

Apesar de serem parecidas, a Figura 4.12 se assemelha mais ao resultado encontrado na aplicação dos algoritmos determinísticos e existem mais sobreposições de trajetórias. No entanto, ainda é visível a obtenção de algumas poucas trajetórias ruins, inclusive a que passa pelo defeito grande, obtida também no GA, e outras atravessando regiões de heterogeneidades mais pronunciadas.

Figura 4.12 - Resultado do mapeamento do pulso ultrassônico no exemplo 2 (seção retangular) para o algoritmo *Ant Colony System* (ACS).



Fonte: Autor.

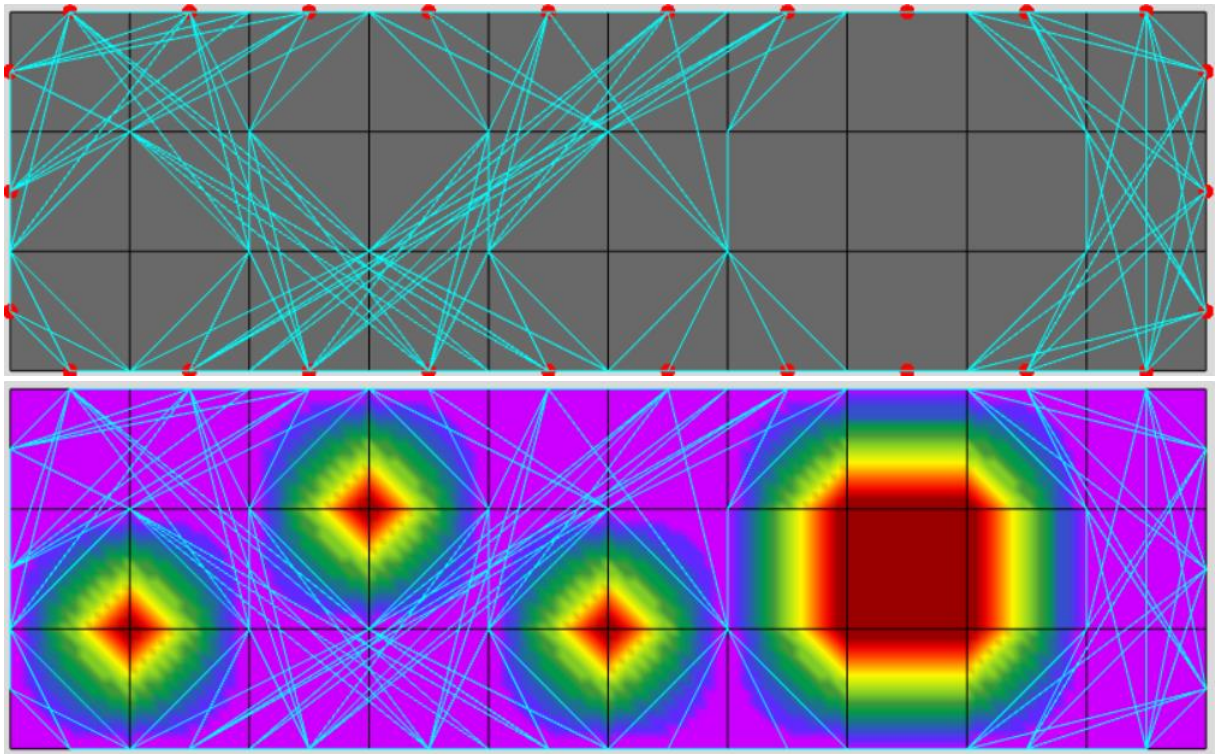
Mesmo com a presença de algumas trajetórias ruins nos métodos bioinspirados, de natureza probabilística e aproximada, considera-se que o pulso foi mapeado com sucesso mais uma vez e os resultados encontram-se dentro do esperado, já que a tendência de desvio dos danos é clara nas Figuras 4.10, 4.11 e 4.12.

Aplicando o algoritmo de Dijkstra à malha mais pobre do exemplo 2 (apenas os 44 nós principais), assim como foi feito no exemplo 1, têm-se a Figura 4.13. De acordo com a Tabela 4.13, o erro médio obtido foi de apenas 1,37% quando comparado com a aplicação do mesmo algoritmo na malha de maior refinamento (3131 nós), quase o dobro do valor encontrado no exemplo 1 (0,79%), o que é esperado para uma seção com maior área danificada. O erro máximo obtido também foi muito baixo, de 3,00%, e novamente é superior ao do exemplo 1, de 2,45%.

Enquanto isso, o erro mínimo foi de 0,00%, encontrado em apenas 10 trajetórias (4,37% do total de 229). Estas 10 trajetórias coincidem com o número encontrado com a hipótese de trajetórias retilíneas, indicando que, assim como no exemplo 1, a aplicação do algoritmo de Dijkstra na malha menos refinada encontrou as trajetórias ótimas apenas quando estas eram retilíneas.

O número de trajetórias ótimas (10) também é notadamente menor que o número encontrado no exemplo 1 (60). O ocorrido deve-se provavelmente ao fato da seção do exemplo 2 possuir mais danos. No geral, observa-se que os valores de erro obtidos foram muito baixos, demonstrando que a utilização do algoritmo de Dijkstra na malha menos refinada é uma alternativa rápida de mapeamento do pulso e que fornece ótimos resultados aproximados.

Figura 4.13 - Resultado do mapeamento do pulso ultrassônico no exemplo 2 (seção retangular) usando o algoritmo de Dijkstra na malha menos refinada (44 nós).



Fonte: Autor.

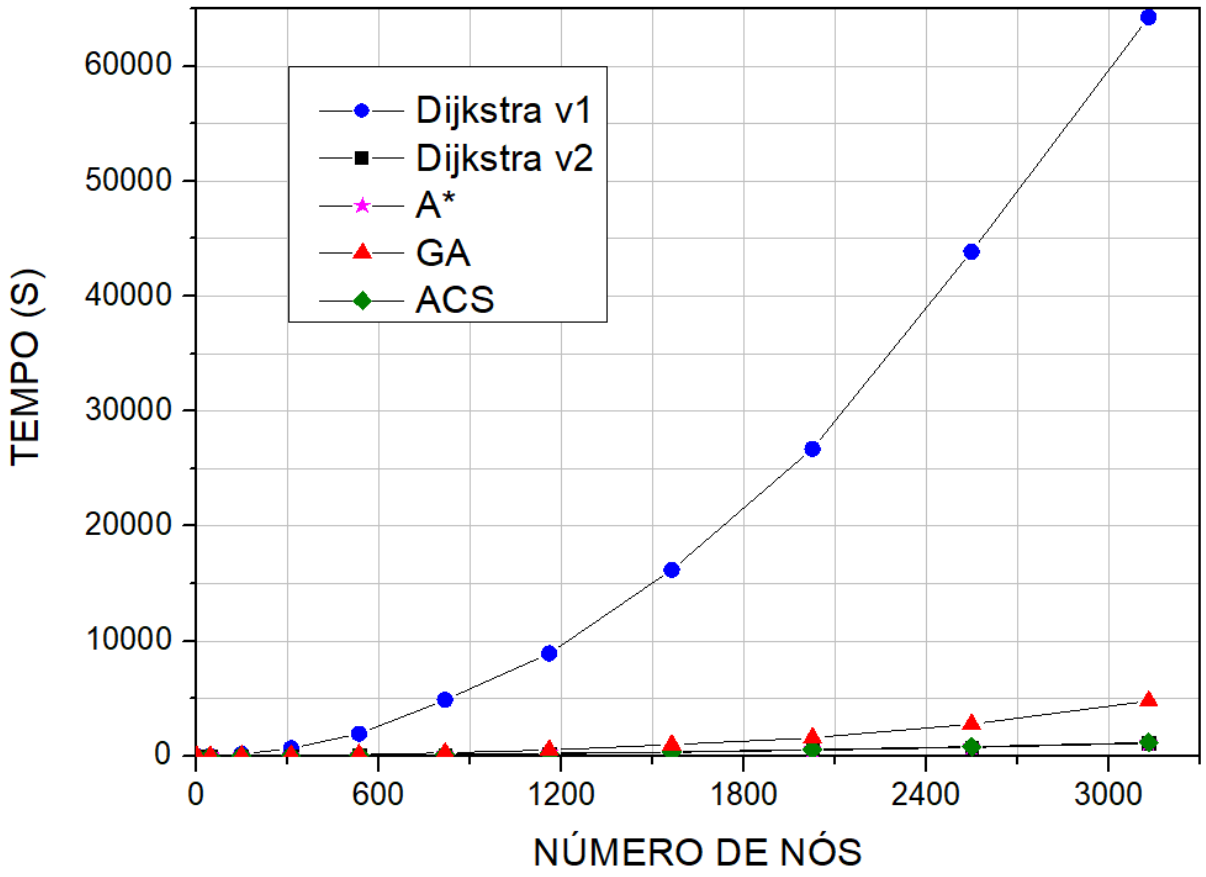
Tabela 4.13 – Erros no exemplo 2 considerando mapeamento com o algoritmo de Dijkstra na malha menos refinada (44 nós).

Erros no exemplo 2 - Algoritmo de Dijkstra	
Número de trajetórias	229
Número de nós da malha	44 (mínimo)
Erro médio	1,37%
Erro máximo	3,00%
Erro mínimo	0,00%
Nº de trajetórias com erro mínimo	10 (40% do total)
Desvio padrão	0,82%

Fonte: Autor.

Elaborando o gráfico de tempo de processamento (em segundos) pelo número de nós da malha da seção para todos os algoritmos, tem-se a Figura 4.14. Agora, a escala horizontal engloba um total de 3131 nós no nível mais alto de refinamento da malha (nível 10). Essa malha é maior que a do exemplo 1, que possuía 2601 nós no nível mais refinado. Houve um aumento de 530 nós (cerca de 20%).

Figura 4.14 – Gráfico comparativo dos tempos de processamento de todos os algoritmos (exemplo 2).



Fonte: Autor.

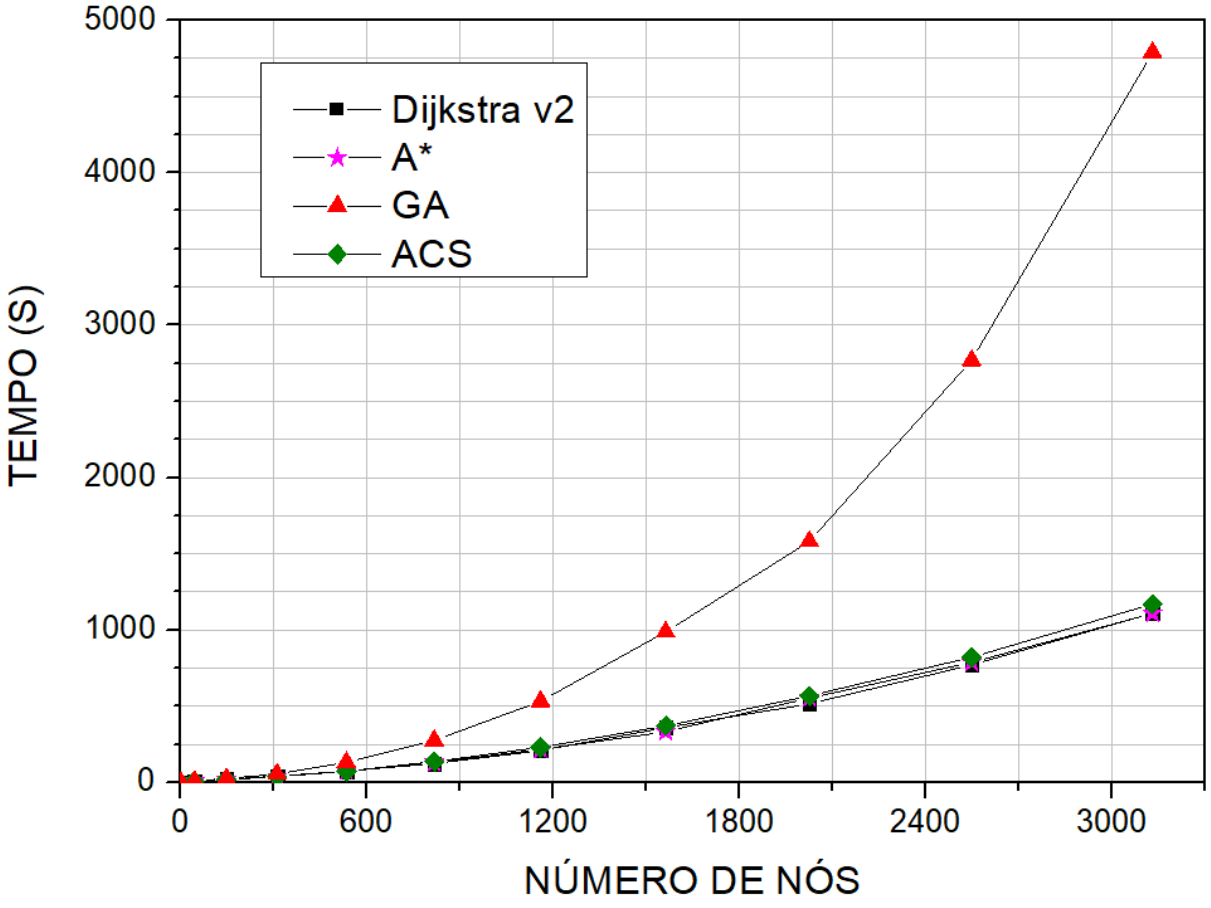
Analisando a Figura 4.14, conclui-se que o resultado obtido no exemplo 1 se repete. O algoritmo Dijkstra v1 apresenta desempenho significativamente inferior se comparado aos outros, atingindo tempos de quase 65.000 segundos (cerca de 18 horas) para o maior nível de refinamento de malha, enquanto o maior tempo dentre os algoritmos restantes é de menos de 5.000 segundos (cerca de 1 hora e 20 minutos).

Este comportamento já era esperado, já que esta é uma implementação inicial, anterior a este trabalho, e os motivos do aumento acentuado do tempo de execução de Dijkstra v1 em relação aos outros algoritmos já foram explorados no exemplo 1.

O tempo de execução de Dijkstra v1 no exemplo 2 também é muito superior ao do exemplo 1, subindo de 20.000 para 65.000 segundos, ou seja, um aumento de 20% na malha de nós gerou um aumento de 225% no tempo de processamento do algoritmo. Este aumento comprova a grande dependência do algoritmo de Dijkstra em relação ao tamanho da malha de nós, também já explorado no exemplo 1.

Como o alto tempo de processamento da implementação Dijkstra v1 não permite a visualização adequada dos tempos de execução dos outros algoritmos, este foi excluído do gráfico e a escala vertical (tempo de processamento em segundos) foi reajustada, dando origem à Figura 4.15.

Figura 4.15 - Gráfico comparativo dos tempos de processamento de todos os algoritmos, exceto Dijkstra v1 (exemplo 2).



Fonte: Autor.

Observando a figura, é imediata a conclusão de que o GA mais uma vez se destaca negativamente, sendo o algoritmo de pior desempenho depois de Dijkstra v1. Nas 3 malhas de nós iniciais, todos os algoritmos apresentam tempos de execução muito próximos, com pontos que praticamente se sobrepõem (escala dificulta a

visualização). Já para a malha mais refinada, com 3131 nós, o GA chega a apresentar tempos de execução de pouco mais de 4750 segundos (cerca de 1 hora e 20 minutos), enquanto os outros algoritmos apresentaram tempos muito próximos, na faixa de 1150 segundos (cerca de 19 minutos), uma diferença considerável, ainda que muito menor que a diferença entre estes algoritmos e a implementação inicial Dijkstra v1.

Os possíveis motivos para a diferença de tempos entre GA e os outros algoritmos já foram explorados no exemplo 1: possível inadequação da estrutura do GA escolhida ao problema das trajetórias, modelo de criação de indivíduos ineficiente e/ou operadores de cruzamento e mutação computacionalmente custosos. Novamente, a utilização do algoritmo genético não se justificou no exemplo em questão, já que ele tolera soluções com até 10% de erro médio e ainda apresenta tempos de processamento maiores que os das outras alternativas propostas.

Eliminando o GA do gráfico e reajustando a escala vertical, de forma análoga ao que foi feito anteriormente, temos como resultado a Figura 4.16. Mais uma vez, os algoritmos Dijkstra v2, A* e ACS despontam como as melhores opções na resolução do problema das trajetórias, com desempenhos semelhantes.

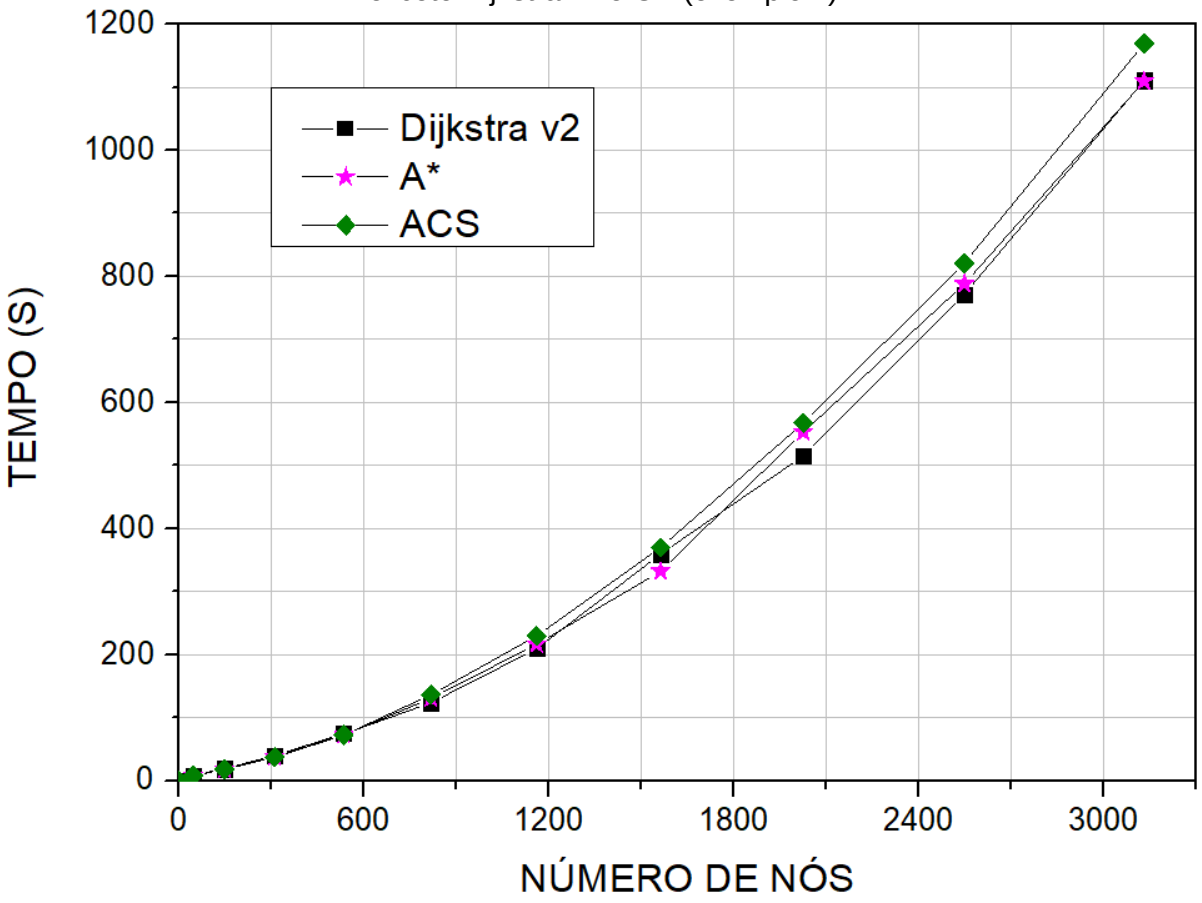
Analisando a figura, é possível confirmar o que foi visto nas figuras anteriores, constatando que os algoritmos apresentam tempos de execução muito pequenos e próximos, entre 1100 e 1200 segundos (cerca de 18 a 20 minutos) para a malha de maior refinamento (3131 nós). Nos níveis de refinamento da malha 1 a 4 (até cerca de 600 nós), os tempos dos algoritmos são muitos semelhantes e os pontos praticamente se sobrepõem (escala dificulta a visualização).

À partir do nível 5, é possível enxergar diferenças entre eles: nos níveis 5 e 6, o algoritmo de Dijkstra é o mais rápido, seguido por A* e pelo ACS. No nível de refinamento 7, as posições de Dijkstra e A* invertem-se, enquanto ACS mantém a terceira posição. Nos níveis de refinamento 8 e 9, ocorre o mesmo que nos níveis 5 e 6 (Dijkstra mais rápido, seguido por A* e ACS). Por fim, no nível 10 da malha de nós, os tempos de Dijkstra e A* praticamente se sobrepõem, enquanto a diferença para o ACS, mais uma vez o mais lento entre os remanescentes, parece aumentar.

Tendo como referência a interpretação visual do gráfico, o algoritmo de Dijkstra é mais rápido em um número maior de instâncias do problema, assim como ocorreu no exemplo 1. No entanto, agora ele é seguido de muito perto por A*, que também se mostra uma ótima alternativa, diferentemente do que ocorreu no exemplo 1, onde a segunda melhor alternativa encontrada foi o algoritmo ACS. Visando esclarecer as

melhores alternativas, ainda incertas, análises utilizando tabelas serão realizadas posteriormente. Os algoritmos GA e Dijkstra v1 foram o quarto e o quinto melhores, respectivamente, em ambos exemplos, sendo possível extrapolar seus desempenhos ruins para outras malhas com números maiores de nós.

Figura 4.16 - Gráfico comparativo dos tempos de processamento de todos os algoritmos, exceto Dijkstra v1 e GA (exemplo 2).



Fonte: Autor.

Compilando os resultados das Figuras 4.14, 4.15 e 4.16, elaborou-se a Tabela 4.14, tornando possível a análise os dados em maior detalhe. A melhor alternativa para cada nível de malha (menor tempo) é destacada na cor vermelha.

À partir da tabela, pode-se perceber que o algoritmo de Dijkstra foi superior aos demais em 4 instâncias do problema, nos níveis 5, 6, 8 e 9 de refinamento da malha de nós. O algoritmo A* também foi superior em 4 instâncias do problema, nos níveis 1, 2, 7 e 10 da malha de nós. O algoritmo ACS foi o mais rápido em 2 instâncias, apenas nos níveis 3 e 4.

Tabela 4.14 – Tempo total dos algoritmos utilizados no exemplo 2, em função do tamanho da malha de nós da seção.

Nível de refinamento da malha	Número de nós	TEMPO TOTAL DO ALGORITMO (seg.)				
		Dijkstra v1	Dijkstra v2	A*	GA 10%	ACS 1%
1	44	18,622	9,084	8,212	8,937	8,352
2	147	158,311	19,324	18,131	25,396	18,646
3	310	678,128	40,272	38,130	54,607	37,837
4	533	1927,986	74,696	73,293	132,394	72,526
5	816	4869,824	123,637	130,841	275,693	136,533
6	1159	8933,849	209,583	216,635	531,608	229,680
7	1562	16196,882	359,722	332,424	987,109	370,026
8	2025	26721,560	515,441	552,867	1581,311	567,487
9	2548	43862,617	771,312	787,913	2768,552	820,983
10	3131	64260,425	1110,595	1109,516	4789,938	1168,959

Fonte: Autor.

Analisando a tabela, nota-se que existe uma igualdade maior entre os algoritmos de Dijkstra e A* do que a visível na Figura 4.16, provavelmente devido à dificuldade de se enxergar o resultado de tempo de processamento em malhas menores. Isso pode ser concluído à partir do fato de que ambos algoritmos foram os melhores em 4 instâncias do exemplo 2. No entanto, enxerga-se uma tendência de superioridade de Dijkstra v2 em malhas maiores, já que este é superior em níveis de malha acima de 5. Uma conclusão mais assertiva pode ser tirada através da comparação das melhores alternativas aos pares.

Novamente, o algoritmo ACS se mostrou competitivo ao longo de todos os níveis de refinamento da malha de nós, apresentando grande potencial. No entanto, foi o melhor algoritmo em apenas 2 instâncias do exemplo 2. No exemplo 1, este havia sido superior nos 4 níveis iniciais da malha. Este posto foi tomado pelo algoritmo A*, que ganhou espaço no exemplo 2: enquanto no exemplo 1 ele havia sido superior em apenas 1 instância, agora foi superior em 4.

Visando esclarecer a diferença de desempenho entre os algoritmos Dijkstra v2 e A* e escolher a melhor alternativa para a resolução do problema das trajetórias, a ser implementada como ferramenta padrão no *software* TUSom, elaborou-se a Tabela 4.15. Quando o primeiro é mais rápido, a diferença é destacada na cor verde e utiliza-

se o sinal negativo. Caso contrário, utiliza-se o sinal positivo e a informação é vermelha.

Tabela 4.15 – Comparação entre os tempos dos algoritmos Dijkstra v2 e A* no exemplo 2.

Refinamento da malha	Número de nós	Tempo Dijkstra v2 (seg.)	Tempo A* (seg.)	Diferença
1	44	9,084	8,212	**+ 10,62%**
2	147	19,324	18,131	**+ 6,58%**
3	310	40,272	38,13	**+ 5,62%**
4	533	74,696	73,293	**+ 1,91%**
5	816	123,637	130,841	**- 5,51%**
6	1159	209,583	216,635	**- 3,26%**
7	1562	359,722	332,424	**+ 8,21%**
8	2025	515,441	552,867	**- 6,77%**
9	2548	771,312	787,913	**- 2,11%**
10	3131	1110,595	1109,516	**+ 0,10%**

Fonte: Autor.

Observando a Tabela 4.15, é possível tirar uma conclusão melhor embasada: o algoritmo de Dijkstra se mostra inferior ao A* em 6 níveis de refinamento de malha e superior em apenas 4. Apesar disso, a diferença continua sendo mínima: no nível de refinamento 10, o algoritmo de Dijkstra é apenas 0,10% mais lento que A*. Caso nessa malha ocorresse o contrário, o algoritmo de Dijkstra provavelmente seria a melhor opção a ser adotada, já que apresentaria melhor desempenho nas malhas mais refinadas, que representam trajetórias do pulso mais próximas das reais. Também é possível notar uma superioridade consistente do algoritmo A* em malhas menores, já que este é mais rápido nos 4 primeiros níveis de refinamento da malha (até cerca de 500 nós). Enquanto isso, Dijkstra v2 apresenta tendência de desempenho melhor em malhas de nós mais refinadas.

As maiores diferenças entre ambos ocorreram nos níveis de refinamento 1 da malha, onde o algoritmo de Dijkstra foi 10,62% mais lento que o seu concorrente, e 8, onde ele foi 6,77% mais rápido que A*. Como o algoritmo A* foi mais rápido que Dijkstra v2 em um número maior de instâncias, é possível concluir que o custo computacional de cálculo da heurística do primeiro algoritmo foi mais vantajoso que o maior número de visitas nodais do segundo no exemplo em questão, ainda que a diferença total dos tempos entre os algoritmos seja pequena.

Como no exemplo 1 o algoritmo A* era superior em apenas 1 instância quando comparado com Dijkstra v2 e agora é superior em 6, é possível que o cálculo da heurística se torne mais vantajoso conforme o número de nós da seção ou conforme a área danificada aumentam. No entanto, como o algoritmo Dijkstra v2 foi superior no primeiro exemplo, apresenta uma tendência de desempenho melhor em malhas de nós mais refinadas em ambos exemplos e é muito pouco inferior na malha de nível 10 do exemplo 2, será eleito de forma definitiva como a melhor alternativa na resolução do problema das trajetórias.

Para escolher definitivamente a segunda melhor alternativa, novamente A* e ACS serão comparados isoladamente, como foi feito no exemplo 1. Agrupando os seus tempos de processamento para diferentes níveis de refinamento da malha de nós, montou-se a Tabela 4.16.

Tabela 4.16 – Comparação entre os tempos dos algoritmos A* e ACS no exemplo 2.

Refinamento da malha	Número de nós	Tempo A* (seg.)	Tempo ACS (seg.)	Diferença
1	44	8,212	8,352	**- 1,68%**
2	147	18,131	18,646	**- 2,76%**
3	310	38,130	37,837	**+ 0,77%**
4	533	73,293	72,526	**+ 1,06%**
5	816	130,841	136,533	**- 4,17%**
6	1159	216,635	229,680	**- 5,68%**
7	1562	332,424	370,026	**- 10,16%**
8	2025	552,867	567,487	**- 2,58%**
9	2548	787,913	820,983	**- 4,03%**
10	3131	1109,516	1168,959	**- 5,09%**

Fonte: Autor.

Analisando a tabela, nota-se que A* é superior em 8 instâncias do exemplo 2, nos níveis 1, 2 e 5 a 10 da malha de nós. Enquanto isso, A* é mais rápido nos níveis 3 e 4. A maior diferença positiva entre ambos ocorre no nível 4, quando A* é 1,06% mais lento que ACS. A maior diferença negativa ocorre no nível 8, quando A* é 10,16% mais rápido que ACS.

Comparando as diferenças de tempo, vê-se que, quando A* é superior, essa diferença é maior do que quando ele é inferior ao ACS. Por este motivo, somado ao

fato de A* sempre fornecer soluções ótimas, de apresentar uma tendência de desempenhos melhores em malhas maiores e devido à sua superioridade clara no exemplo 2, que possui maiores quantidade de danos, área danificada e malha com maior número de nós, consequentemente um exemplo mais complexo, optou-se pela sua escolha como a segunda melhor opção na resolução do problema das trajetórias.

Dessa forma, após o teste nos dois exemplos, com os parâmetros e as estruturas adotados para cada algoritmo, as melhores opções para a resolução do problema das trajetórias foram, nesta ordem: Dijkstra v2, A*, ACS, GA e Dijkstra v1. Com relação ao exemplo 1, as posições de A* e ACS foram trocadas e as posições restantes permaneceram inalteradas. Com isso, o algoritmo Dijkstra v2 será implementado no *software* TUSom como alternativa padrão para a determinação da trajetória de pulsos ultrassônicos em estruturas de concreto, dado um mapa de velocidades conhecido.

Novamente, o algoritmo escolhido como melhor alternativa (Dijkstra v2) será comparado aos restantes (ACS, GA e Dijkstra v1), com o intuito de visualizar as diferenças relativas de tempos entre eles. Comparando Dijkstra v2 com o algoritmo ACS, elaborou-se a Tabela 4.17 de forma análoga às Tabelas 4.15 e 4.16, acrescentando os parâmetros variáveis do algoritmo: número de vizinhos adotados na CL e número de iterações. Os valores dos outros parâmetros já foram definidos na seção 3.4 ($m = 10$, $\beta = 2$, $\xi = \rho = 0{,}1$, $q_0 = 0{,}5$ e $\tau_0 = 1/n.C^{nn}$).

Tabela 4.17 - Comparação entre os tempos dos algoritmos ACS e Dijkstra v2 no exemplo 2.

Refinamento da malha	Número de nós	Nº de vizinhos	Nº de iterações	Erro médio	Tempo ACS (seg.)	Tempo Dijkstra v2 (seg.)	Diferença
1	44	2	5	0,67%	8,352	9,084	**- 8,06%**
2	147	2	5	0,90%	18,646	19,324	**- 3,51%**
3	310	3	5	0,97%	37,837	40,272	**- 6,05%**
4	533	5	20	0,96%	72,526	74,696	**- 2,91%**
5	816	5	80	1,00%	136,533	123,637	**+ 10,43%**
6	1159	6	100	0,99%	229,680	209,583	**+ 9,59%**
7	1562	6	200	1,00%	370,026	359,722	**+ 2,86%**
8	2025	9	190	0,99%	567,487	515,441	**+ 10,10%**
9	2548	10	320	1,00%	820,983	771,312	**+ 6,44%**
10	3131	11	900	1,00%	1168,959	1110,595	**+ 5,26%**

Fonte: Autor.

Também calculou-se o erro médio obtido em cada nível de refinamento da malha de nós. Assim como na Tabela 4.6, este erro é obtido comparando-se os tempos de viagem dos pulsos ultrassônicos entre ACS e Dijkstra v2 para um mesmo nível de refinamento da malha de nós.

A maior diferença positiva entre os algoritmos ocorre no nível 5 da malha de nós (1562 nós), quando ACS é 10,43% mais devagar que Dijkstra v2. Essa diferença é pouco menor quando comparada com a maior diferença positiva entre Dijkstra v2 e A* (o segundo melhor algoritmo), de 10,62%. A maior diferença negativa ocorre no nível 1 de malha, quando ACS é 8,06% mais rápido. Com isso, vemos que ACS se destaca especialmente nas menores malhas, sendo superior a Dijkstra v2 nas 4 primeiras instâncias do exemplo 2, assim como ocorreu no exemplo 1. A partir do nível de refinamento 5 da malha, Dijkstra v2 é sempre superior.

Novamente, os parâmetros variáveis do ACS influenciaram mais a sua execução que o tamanho da malha de nós: apenas 10 formigas (valor fixado), 11 vizinhos e 900 iterações foram suficientes para encontrar trajetórias com erro médio de 1,00% na maior das malhas (3131 nós).

Diferentemente do que ocorreu no exemplo 1, onde a quantidade mínima de vizinhos e iterações (2 e 5, respectivamente) foram suficientes para resolver o exemplo com erro menor que 0,5% em todos os níveis de malha, agora o número de vizinhos e iterações subiu bastante. Isso se deve à maior área danificada, número de nós e complexidade geral do exemplo 2, confirmando a hipótese levantada no exemplo 1. Comparando o primeiro e último níveis da malha de nós, o número de vizinhos na CL aumentou de 2 para 11 (aumento de 450%), enquanto o número de iterações aumentou de 5 para 900 (aumento de 17.900%).

Quando analisamos a qualidade das trajetórias obtidas pelo algoritmo ACS no nível 10 da malha de nós, vide Tabela 4.18, o maior erro obtido em uma única trajetória foi de 8,18%, um valor relativamente baixo, dada a complexidade do exemplo 2. Além disso, o erro mínimo obtido, de 0,00%, representa os casos em que as trajetórias ótimas foram encontradas pelo algoritmo. A diferença é que, agora, este erro foi encontrado em 35 trajetórias (para as trajetórias retilíneas haviam sido 10), número maior e que representa 15,28% do total de trajetórias (229). Isso indica que o ACS é capaz de encontrar trajetórias ótimas mesmo quando estas não são retilíneas, um sinal de seu bom desempenho.

Tabela 4.18 – Erros no exemplo 2 considerando mapeamento com o algoritmo ACS na malha mais refinada (3131 nós).

Erros no exemplo 2 - Algoritmo ACS	
Número de trajetórias	229
Número de nós da malha	3131 (máximo)
Erro médio	1,00%
Erro máximo	8,18%
Erro mínimo	0,00%
Nº de trajetórias com erro mínimo	35 (15,28% do total)
Desvio padrão	1,56%

Fonte: Autor.

Ainda na malha de nós mais refinada, o desvio padrão das medidas de tempo foi de 1,56% (56% maior que ao erro médio), indicando um conjunto de dados relativamente uniforme e que varia pouco em relação à média. Dessa forma, conclui-se que o algoritmo ACS encontra respostas com ótima qualidade, ou seja, pontos subótimos com erros baixos na maioria das trajetórias. Isso é realizado em tempos bastante satisfatórios, que competem na mesma ordem de grandeza com os melhores algoritmos. Assim, a sua aplicação inicial foi bastante promissora.

Apesar disso, o ACS não foi o algoritmo mais rápido e fornece soluções aproximadas (erro médio de até 1%). Dessa forma, seu uso não se justificou na resolução do problema das trajetórias para os exemplos 1 e 2 (para as estruturas do algoritmo implementadas e com os parâmetros adotados).

Com o intuito de enxergar também a variação dos parâmetros do GA, elaborou-se a Tabela 4.19, de maneira análoga à anterior, comparando o algoritmo Dijsktra v2 e o GA. Agora, os parâmetros exibidos são o número de indivíduos e de gerações adotados para o GA em cada nível de refinamento da malha, com tolerância de erro médio de até 10%. Os outros parâmetros do algoritmo foram definidos na seção 3.3 (PC = 0,9, PA = 0,1 e 2 elites por geração). Assim como na Tabela 4.17, o erro médio é obtido comparando-se os tempos de viagem dos pulsos ultrassônicos entre o GA e o algoritmo Dijkstra v2 para um mesmo nível da malha de nós.

No que diz respeito ao tempo de processamento dos algoritmos, já era claro o desempenho significativamente inferior do GA. No entanto, a dimensão destas diferenças não era evidente. Surpreendentemente, o GA é superior a Dijkstra v2 em 1 das instâncias do exemplo 2, o nível 1 da malha de nós, em que é 1,62% mais

rápido. À partir deste nível, a diferença entre ambos cresce de forma acentuada e o GA chega a ser 331,29% mais lento na maior malha de nós (3131 nós).

Assim como o ACS, o GA também depende em maior proporção dos seus parâmetros variáveis do que do tamanho da malha de nós. Na Tabela 4.19, estes parâmetros variaram ainda menos que no ACS. O número de indivíduos foi de 50 a 120 da malha mais pobre para a mais refinada. Enquanto isso, o número de gerações evoluiu de 15 para 25 neste mesmo intervalo. Ou seja, o número de indivíduos aumentou 140% e o de gerações aumentou 66,67%, enquanto o número de nós da malha aumentou cerca de 71 vezes (de 44 para 3131 nós), assim como no exemplo 1.

Tabela 4.19 - Comparação entre os tempos dos algoritmos GA e Dijkstra v2 no exemplo 2.

Refinamento da malha	Número de nós	Indivíduos	Gerações	Erro médio	Tempo GA (seg.)	Tempo Dijkstra v2 (seg.)	Diferença
1	44	50	15	9,56%	8,937	9,084	**- 1,62%**
2	147	50	15	8,23%	25,396	19,324	**+ 31,42%**
3	310	50	20	8,67%	54,607	40,272	**+ 35,60%**
4	533	70	20	8,95%	132,394	74,696	**+ 77,24%**
5	816	80	20	9,25%	275,693	123,637	**+ 122,99%**
6	1159	90	20	9,27%	531,608	209,583	**+ 153,65%**
7	1562	100	20	9,82%	987,109	359,722	**+ 174,41%**
8	2025	100	20	9,92%	1581,311	515,441	**+ 206,79%**
9	2548	120	20	8,02%	2768,552	771,312	**+ 258,94%**
10	3131	120	25	9,87%	4789,938	1110,595	**+ 331,29%**

Fonte: Autor.

Quando comparado ao exemplo 1, alguns números são iguais: ambos apresentaram 120 indivíduos e 25 gerações na malha de nós mais refinada. Apesar de ser um dado que causa estranhamento, já que o exemplo 2 possui mais danos e uma malha de nós maior, essa igualdade se justifica devido ao erro médio menor tolerado no exemplo 1, de 5%, enquanto no exemplo 2 é de 10%, o dobro.

Quando trata-se da qualidade das trajetórias obtidas pelo GA, vide Tabela 4.20, o erro médio foi relativamente baixo, de 9,87%. No entanto, assim como no exemplo 1, algumas trajetórias obtidas foram muito ruins, problema exacerbado pelo exemplo mais complexo. A trajetória mapeada com maior erro é 515,36% mais lenta que a ótima (no exemplo 1 era 102,17%). Acompanhando também a tendência do exemplo

1, este erro extremo foi um caso pontual, já que a segunda trajetória de maior erro apresenta 130,15% de diferença em relação à ótima (no exemplo 1 era de 34,16%). Ambos valores são consideravelmente superiores ao erro máximo obtido com a hipótese de trajetórias retilíneas (34,55%).

Tabela 4.20 – Erros no exemplo 2 considerando mapeamento com o GA na malha mais refinada (3131 nós).

Erros no exemplo 1 - GA	
Número de trajetórias	229
Número de nós da malha	3131 (máximo)
Erro médio	9,87%
Erro máximo	515,36%
Erro mínimo	0,00%
Nº de trajetórias com erro mínimo	1 (0,44% do total)
Desvio padrão	35,95%

Fonte: Autor.

Assim como no exemplo 1, quando se utilizou o mapeamento com o GA, o erro médio das trajetórias foi menor que o das trajetórias retilíneas, mas o erro máximo foi notavelmente superior. Conclui-se então que o erro médio do GA é composto por muitas trajetórias de erros baixos e acaba sendo elevado por erros pontuais muito grandes, comportamento que provavelmente se repetiria em novos exemplos. Possíveis soluções para estes erros extremos foram discutidas no exemplo 1.

Novamente, a trajetória de erro mínimo encontrada foi de 0,00%. No entanto, esse valor foi obtido apenas em 1 trajetória (0,44% do total). No exemplo 1, o erro de 0,00% foi obtido em 7 trajetórias (cerca de 5% do total), um número maior, esperado pela menor complexidade do exemplo. Observando estes resultados, é clara a tendência do GA de encontrar respostas ótimas apenas para um número pequeno de trajetórias.

O desvio padrão das medidas foi de 35,95%, o que indica um conjunto de dados menos uniforme que o obtido no ACS (enquanto neste o desvio padrão era 56% maior que o erro médio tolerado, no GA ele é 260% maior). Comparando com o exemplo 1, o desvio padrão do ACS diminuiu, enquanto o do GA aumentou, não sendo possível observar nenhuma tendência. No entanto, é clara a observação de que grandes erros pontuais elevam a média de erros e o desvio padrão do GA.

A implementação inicial Dijkstra v1 também foi comparada com o algoritmo Dijkstra v2. Este resultado foi reunido na Tabela 4.21. Como ambos algoritmos são determinísticos, fornecem os tempos de referência e não existe erro envolvido.

Tabela 4.21 – Comparação entre os tempos dos algoritmos Dijkstra v1 e v2 no exemplo 2.

Refinamento da malha	Número de nós	Tempo Dijkstra v1 (seg.)	Tempo Dijkstra v2 (seg.)	Diferença
1	44	18,622	9,084	**+ 105,00%**
2	147	158,311	19,324	**+ 719,25%**
3	310	678,128	40,272	**+ 1583,87%**
4	533	1927,986	74,696	**+ 2481,11%**
5	816	4869,824	123,637	**+ 3838,81%**
6	1159	8933,849	209,583	**+ 4162,68%**
7	1562	16196,882	359,722	**+ 4402,61%**
8	2025	26721,560	515,441	**+ 5084,21%**
9	2548	43862,617	771,312	**+ 5586,75%**
10	3131	64260,425	1110,595	**+ 5686,13%**

Fonte: Autor.

A diferença exorbitante entre os algoritmos implementados e Dijkstra v1, já observada na Figura 4.14, é reforçada pela tabela. No primeiro nível de refinamento de malha, Dijkstra v1 já apresenta mais que o dobro do tempo de processamento de Dijkstra v2 (é 105% mais lento). A diferença de tempos cresce exponencialmente conforme a malha aumenta: no nível 10 da malha de nós, o tempo de Dijkstra v1 é 5686,13% maior que o de Dijkstra v2.

Finalizada a comparação de tempos de processamento entre os algoritmos e escolhidas as melhores opções para a resolução do problema das trajetórias no *software* TUSom, o tempo de pré-processamento da melhor alternativa (Dijkstra v2) foi novamente comparado ao tempo total do algoritmo. O resultado desta comparação se encontra na Tabela 4.22.

Observa-se na tabela que, enquanto na malha de menor refinamento (44 nós) o pré-processamento representa apenas 1,83% do tempo total do algoritmo, sendo praticamente 53 vezes menor que o seu tempo de aplicação, na malha de nós com maior refinamento (3131 nós), o tempo de pré-processamento representa 76,08% do tempo total do algoritmo, sendo 3 vezes maior que o seu tempo de aplicação. Estes

números são muito semelhantes aos obtidos no exemplo 1, de 1,80% na menor malha e 80,13% na maior.

Tabela 4.22 – Comparação dos tempos de pré-processamento e de aplicação do algoritmo Dijkstra v2 no exemplo 2.

Refinamento da malha	Número de nós	Tempo pré-process. (seg.)	Tempo algoritmo (seg.)	Tempo total (seg.)	Parcela do todo (pré-process.)
1	44	0,150	8,062	8,212	1,83%
2	147	1,846	16,285	18,131	10,18%
3	310	8,188	29,942	38,130	21,47%
4	533	24,419	48,874	73,293	33,32%
5	816	57,972	72,869	130,841	44,31%
6	1159	115,166	101,469	216,635	53,16%
7	1562	202,564	129,860	332,424	60,94%
8	2025	378,627	174,24	552,867	68,48%
9	2548	575,726	212,187	787,913	73,07%
10	3131	844,107	265,409	1109,516	76,08%

Fonte: Autor.

Assim, novamente observa-se que o tempo necessário à etapa de pré-processamento do algoritmo cresce em um ritmo muito maior que o da aplicação do algoritmo em si, passando a ser maior que este quando a malha possui cerca de 1030 nós. À partir das observações, é possível afirmar que em malhas maiores o pré-processamento irá ocupar parcelas cada vez maiores do tempo total de processamento. Apesar disso, ele aparenta apresentar um comportamento assintótico, ou seja, as parcelas do tempo total consumidas pelo pré-processamento aumentam de um nível de malha para outro, mas em proporção cada vez menor.

5 CONCLUSÕES

A hipótese de trajetórias retilíneas (Figuras 4.1 e 4.9) gerou erros máximos da faixa de 30%, valores elevados, reforçando a importância do mapeamento dos pulsos. Como esperado, o erro foi superior no exemplo 2 devido à maior área danificada da seção transversal. Ainda para as trajetórias retilíneas, o erro nulo foi obtido em 40% das trajetórias do exemplo 1 e em 4,37% das trajetórias do exemplo 2, indicando que nestas a trajetória retilínea coincide com a melhor encontrada pelos algoritmos determinísticos.

Essas diferenças devem-se à menor área danificada da seção transversal no exemplo 1. No exemplo 2, o comportamento é mais próximo do esperado em uma seção experimental, que possui trechos de heterogeneidade intrínsecos ao material concreto, por mais bem dosado que seja. Essa heterogeneidade faz com que seja muito pouco provável que um pulso siga uma trajetória totalmente retilínea. Além disso, a seção transversal do exemplo 2 possui uma parcela maior de área danificada, o que faz com que os pulsos desviem das heterogeneidades e pouco provavelmente um pulso retilíneo seja ótimo.

Após o processo de determinação das trajetórias do pulso ultrassônico utilizando os algoritmos propostos, é possível concluir que o processo foi bem-sucedido, tomando como referência as Figuras 4.2 a 4.4, referentes ao exemplo 1, e 4.10 a 4.12, referentes ao exemplo 2. Observa-se que, após a aplicação dos algoritmos, os pulsos desviaram das heterogeneidades simuladas na seção transversal, um comportamento mais fiel ao real. Com a hipótese das trajetórias retilíneas, apresentada nas Figuras 4.1 e 4.9, para os exemplos 1 e 2, respectivamente, vemos que o pulso trilhava caminhos muito diferentes no interior das seções, atravessando zonas de heterogeneidade pronunciada, cenário muito distante do real.

Em razão da sua natureza determinística, os algoritmos de Dijkstra e A* fornecem as melhores soluções possíveis para problema das trajetórias, dada uma malha de nós implementada computacionalmente na forma de grafo. Isso não significa que estas respostas coincidiriam perfeitamente com as de um arranjo experimental, já que isso ocorreria apenas quando o número de nós da malha computacional tendesse ao infinito, representando o meio contínuo. No entanto, são uma boa

aproximação, dado o modelo matemático adotado, e são tomadas como referência (possuem erro nulo) para os tempos de viagem do pulso nos exemplos estudados.

Como o erro envolvido é nulo, os algoritmos determinísticos geraram resultados visuais melhores que os algoritmos bioinspirados. Existe grande sobreposição das trajetórias e caminhos preferenciais percorridos nas parcelas de concreto íntegro. Enquanto isso, nas imagens geradas pelos algoritmos bioinspirados, a sobreposição de trajetórias é bem menor e foram encontradas algumas trajetórias ruins, que atravessam os defeitos.

As trajetórias ruins devem-se ao fato dos algoritmos bioinspirados (GA e ACS) trabalharem com tolerâncias de erro médio. Estes valores foram adotados evitando ultrapassar os erros médios gerados pela hipótese das trajetórias retilíneas. Caso isso ocorresse, o mapeamento não faria sentido, pois o *software* apresentaria maior custo computacional para devolver respostas de qualidade inferior às retilíneas, que podem ser obtidas de forma imediata.

Os erros atrelados aos algoritmos bioinspirados são facilmente visualizáveis nas Figuras 4.3 e 4.4 para o exemplo 1, especialmente se comparadas à Figura 4.2, mapeada pelos algoritmos determinísticos. No entanto, como o erro tolerável no exemplo 1 é menor, trajetórias melhores são obtidas, atravessando em alguns casos as regiões de heterogeneidades leves (em azul-escuro e verde nas figuras). No exemplo 2, comparam-se as Figuras 4.11 e 4.12 com a 4.10. Como o erro tolerado é maior, algumas trajetórias obtidas apresentaram grande erro, atravessando áreas com defeitos mais pronunciados (nas cores amarelo e vermelho nas figuras). Apesar das diferenças nas imagens, os resultados obtidos encontram-se dentro do esperado.

Analisando as Figuras 4.5 e 4.13 (exemplos 1 e 2, respectivamente), ainda é possível notar uma nova utilidade dos algoritmos determinísticos. O algoritmo de Dijkstra, por exemplo, pode ser utilizado em malhas de nós menos refinadas, visando um mapeamento mais rápido das trajetórias do pulso ultrassônico em seções de concreto, pois fornece respostas aproximadas de grande qualidade. No entanto, o algoritmo de Dijkstra em malhas mais refinadas só foi capaz de encontrar trajetórias ótimas quando estas são retilíneas. Como se repetiu para os dois exemplos, este comportamento pode ser extrapolado para outras seções.

Também elaboraram-se gráficos do número de nós da malha pelo tempo de processamento dos algoritmos aplicados, em segundos. Para os exemplos 1 e 2, os gráficos são apresentados nas Figuras 4.6 e 4.14, respectivamente, e incluem todos

os algoritmos: a implementação inicial Dijkstra v1 (anterior a este trabalho), os algoritmos determinísticos Dijkstra v2 e A* e os algoritmos bioinspirados GA e ACS.

Em ambas figuras, vê-se uma diferença significativa entre o tempo de execução da implementação inicial Dijkstra v1 e os outros algoritmos. Os principais motivos para o desempenho ruim deste algoritmo estão relacionados à sua implementação, principalmente na etapa da montagem do grafo e em processos de ordenação ineficientes, que atrasam cada iteração. Desta forma, o algoritmo Dijkstra v1 é uma alternativa inferior para a resolução do problema das trajetórias, resultado já esperado por ser uma implementação anterior ao trabalho.

Com o intuito de visualizar a diferença de tempos entre os algoritmos restantes (Dijkstra v2, A*, GA e ACS), elaboraram-se os gráficos das Figuras 4.7 (exemplo 1) e 4.15 (exemplo 2). Observando-as, foi possível concluir que mais uma vez um algoritmo se destaca negativamente: o GA. Os motivos para o resultado negativo são possíveis escolhas ruins de operadores, parâmetros ou estruturas, além da própria natureza do algoritmo, que pode não ser a que melhor se enquadra na resolução do problema das trajetórias.

Como exemplos práticos destes motivos, podem ser citados: o modelo de cruzamento adotado (uniforme), já que este operador possui variações mais simples (cruzamento de um ou dois pontos, por exemplo); O modelo de indivíduo escolhido, que nas maiores malhas faz com que os indivíduos sejam vetores demasiadamente grandes; A probabilidade de mutação, adotada como 10%, que pode estar embaralhando boas soluções ao invés de contribuir para buscas globais (*exploration*), que trazem variabilidade à população, diminuindo as chances de convergência prematura a mínimos locais.

Dessa forma, o GA é o quarto melhor algoritmo para a resolução do problema das trajetórias, fornecendo as respostas com a menor qualidade dentre todas as opções disponíveis e com tempos de processamento superiores a outros algoritmos, a frente apenas da implementação inicial Dijkstra v1. No entanto, o GA é consideravelmente mais rápido que Dijkstra v1, apresentando tempos cerca de 800% menores na malha de nós mais refinada para o exemplo 1 e 1200% menores para o exemplo 2. Além disso, o GA é menos dependente da malha de nós, assim como ocorre no ACS: apenas 120 indivíduos e 25 gerações foram suficientes para resolver o problema das trajetórias dentro do erro estipulado em ambos exemplos.

Também existem muitas possibilidades de estruturas diferentes e o GA é alvo constante de estudos por apresentar desempenhos extremamente satisfatórios na resolução dos mais diversos problemas de otimização. Dessa forma, novas variações estão em constante surgimento e aprimoramento e pode-se considerar que a primeira aplicação do GA não foi em vão. Pelo contrário, ela abre a possibilidade de estudos e aplicação do algoritmo com estruturas diferentes na resolução do problema das trajetórias.

Nas Figuras 4.8 e 4.16 é possível visualizar as diferenças de tempo, ainda que pequenas, entre os algoritmos restantes: Dijkstra v2, A* e ACS. Partindo da observação das figuras, nota-se que os algoritmos são bem parelhos nas instâncias iniciais do problema e, conforme o número de nós da malha aumenta, as diferenças tornam-se mais pronunciadas.

À partir de uma análise visual inicial, Dijkstra aparenta ser o melhor algoritmo, seguido de A* e ACS. Os melhores desempenhos de Dijkstra v2 foram obtidos nas malhas de nós mais refinadas, que constituem situações de maior interesse prático por representarem trajetórias do pulso mais próximas das reais. Usando tabelas, esta análise foi confirmada posteriormente. Enquanto os algoritmos ACS e A* possuem poucas variações de estrutura e parâmetros possíveis, Dijkstra possui a variação bidirecional, que altera significativamente o modo de operar do algoritmo, diminuindo o número de visitas nodais e podendo gerar resultados ainda melhores.

Após novas análises na forma de tabelas, conclui-se que o algoritmo A* apresenta uma tendência de melhor desempenho em malhas de nós mais refinadas, encontra soluções ótimas (erro nulo) e apresentou tempos de execução menores no exemplo 2 (mais complexo) quando comparado ao algoritmo ACS. Por estes motivos, foi adotado como a segunda melhor opção. Dessa forma, a classificação final dos algoritmos toma a seguinte forma, em ordem decrescente de melhores tempo de processamento e qualidade das soluções fornecidas: Dijkstra v2, A*, ACS, GA e Dijkstra v1. Isso implica que o algoritmo Dijkstra v2 será implementado como alternativa padrão da resolução do problema das trajetórias no *software* TUSom.

Apesar de ACS ser o terceiro melhor algoritmo, a sua aplicação foi muito promissora. O ACS é o mais rápido em 4 instâncias do exemplo 1 e em 2 instâncias do exemplo 2, apesar de trabalhar com tolerâncias de erro. Além disso, apresentou pequenas diferenças de tempo de processamento com relação ao melhor algoritmo,

Dijkstra v2, sendo competitivo em todos os níveis de refinamento da malha de nós, mesmo que apresente tendência de melhores desempenhos em malhas menores.

O bom desempenho de ACS deve-se em parte a uma das maiores vantagens dos algoritmos bioinspirados, já citada anteriormente: a menor sensibilidade com relação ao tamanho da malha de nós. Para os algoritmos bioinspirados, a malha não impõe o mesmo ritmo de crescimento de dados que impõe aos algoritmos determinísticos. Além disso, a própria criação da família de algoritmos de otimização por colônia de formigas (ACO) tem como inspiração o encontro dos menores caminhos por formigas na natureza. Desta forma, o algoritmo se adapta muito bem à esse tipo de problema, hipótese comprovada pelo seu desempenho notável no TSP e agora no problema das trajetórias de pulsos ultrassônicos. O grande potencial do ACS também abre precedentes para a aplicação de outras estruturas de ACO, como o *MMAS*, *Rank-based AS* ou *Population-based AS*.

Por fim, foram elaboradas as Tabelas 4.11 e 4.22, onde compararam-se os tempos das etapas de pré-processamento e de aplicação do algoritmo Dijkstra v2, obtendo resultados inusitados. Em ambos exemplos, a taxa de crescimento do tempo de pré-processamento dos algoritmos é bastante acelerada. Dada a proporção deste crescimento, é provável que tempos ainda maiores sejam obtidos em malhas com maior número de nós. Apesar disso, o crescimento do tempo de pré-processamento aparenta tender para um valor limite assintótico. Dessa forma, caso deseje-se aprimorar ainda mais os algoritmos, a prioridade deve ser a etapa de pré-processamento. Nesta etapa, o grafo que representa a malha da seção transversal de concreto é montado.

O aprimoramento desta etapa pode ser feito através de novas representações para os grafos, como listas de adjacência, além de novos métodos de criação de malhas, adaptados para cada campo de velocidades que se deseja estudar. Nesse caso, adotam-se malhas mais refinadas apenas em pontos de maior interesse, onde existem caminhos preferencias dos pulsos ou maiores gradientes de velocidades, por exemplo, reduzindo o número total de nós da malha.

Em resumo, apesar dos algoritmos bioinspirados (GA e ACS), foco central deste trabalho, não terem apresentado o melhor desempenho dentre as alternativas implementadas, conseguiram resolver o problema das trajetórias com tempos de processamento muito menores que a alternativa anterior utilizada pelo *software* TUSom (Dijkstra v1). Além disso o algoritmo ACS apresentou desempenho

competitivo com relação aos melhores algoritmos (Dijkstra v2 e A*). Dessa forma, o ACO, família de algoritmos da qual o ACS faz parte, se mostra como uma alternativa interessante na resolução do problema das trajetórias e abre precedentes para o estudo de variações de sua estrutura.

Os algoritmos determinísticos (Dijkstra v2 e A*), ao contrário do que se presumia inicialmente, dada a sua característica de varrer uma parcela maior do espaço de buscas, resultaram nos menores tempos de processamento. Dessa forma, consistem em ótimas ferramentas para a resolução do problema das trajetórias, com desempenhos muito similares entre si e levemente superiores ao do ACS. Eles foram capazes de diminuir o tempo de determinação das trajetórias ótimas no exemplo 1 da ordem de 5 horas e 30 minutos para menos de 10 minutos e no exemplo 2 de 18 horas para cerca de 20 minutos. Ainda por cima, obtêm trajetórias ótimas, com erro nulo. Assim sendo, pode-se concluir que o objetivo do trabalho foi cumprido com sucesso.

6 SUGESTÕES PARA TRABALHOS FUTUROS

Inúmeras ideias surgiram ao longo deste trabalho, que se desenhou como uma investigação inicial da aplicação dos algoritmos bioinspirados como alternativa de resolução do problema de determinação das trajetórias ótimas de pulsos ultrassônicos. Como versões iniciais de diferentes algoritmos foram propostas, uma das sugestões para trabalhos futuros seria investigar mais a fundo algoritmos que demonstraram potencial. Para os algoritmos que não tiveram um bom desempenho, a melhor estrutura para a abordagem deste problema pode não ter sido encontrada, abrindo margem para essa investigação também.

Tratando dos algoritmos na ordem em que apareceram, o próprio algoritmo de Dijkstra, que revelou-se como a segunda melhor alternativa para o mapeamento dos pulsos ultrassônicos, possui algumas variações de estrutura, como o algoritmo de Dijkstra bidirecional, que poderia ser implementado.

Pensando no algoritmo A*, por ser a alternativa que apresentou os menores tempos de processamento em conjunto com a melhor qualidade das soluções (menores tempos de propagação do pulso possíveis para uma dada malha de nós), podem-se pensar em novos tipos de informação heurística que têm potencial para diminuir o número de nós visitados, encontrando as trajetórias ótimas mais rápido. Existe ainda o chamado algoritmo ALT (A*, *Landmarks* e *Triangle Inequality*) proposto por Goldberg e Harrelson (2004), em que se utilizam pontos de referência previamente calculados na malha para auxiliar no cálculo rápido de heurísticas, diminuindo o número de visitas nodais e tornando o algoritmo mais eficiente. Ainda por cima, por se utilizar da desigualdade triangular, fornece heurísticas consistentes, que produzem sempre soluções ótimas.

Sugere-se também a variação dos parâmetros dos algoritmos bioinspirados (tanto do AG quanto do ACS), buscando valores que se adequem melhor ao problema das trajetórias. O ACS, em especial, tem potencial para superar os resultados gerados pelos algoritmos determinísticos do ponto de vista de tempo de processamento.

Pensando no GA, por exemplo, podem-se variar parâmetros como a probabilidade de cruzamento, que deve variar entre 0,80 e 0,95, e a probabilidade de mutação, que deve assumir valores menores, visto que o valor atual (0,10) se torna elevado conforme o tamanho dos indivíduos aumenta, podendo desordenar boas

soluções encontradas pelo operador de cruzamento ao invés de promover a busca global.

Atualmente, os filhos avançam à próxima geração independentemente de sua avaliação. Pode ser proposto um modelo onde, após a criação dos filhos, sorteia-se aleatoriamente se filhos ou pais serão passados para a próxima geração, visando conservar a diversidade genética, ou avaliar pais e filhos e passar os melhores à próxima geração, buscando aumentar a pressão seletiva sobre os melhores indivíduos. O número de elites, adotado como 2 no trabalho, também pode ser aumentado, visando conservar mais indivíduos bem classificados, o que sacrifica parte da variabilidade genética, mas pode gerar resultados superiores no que diz respeito à velocidade de convergência do método.

Variações de estrutura mais radicais também são possíveis no GA: na seleção, podem ser testados os modelos de seleção por torneio e ranking; No cruzamento, podem ser testados os cruzamentos de dois pontos, o *edge recombination* e o mapeamento parcial, sendo os dois últimos provenientes da tipologia do GA baseada em ordem (LINDEN, 2006); Na mutação, podem ser testados modelos de inversão de sublistas e de permutação de elementos, também provenientes do GA baseado em ordem e menos custosos computacionalmente. Por fim, o operador inver-over, que substitui o cruzamento e mutação, proposto por Michalewicz (2004), pode ser implementado. Seus estudos demonstraram resultados que aliam rapidez de processamento e boa qualidade de soluções em aplicações de GA baseado em ordem. O modelo de geração de indivíduos também pode ser alterado, visando gerar indivíduos menores, que apresentam menos custo computacional de criação e operação.

Como proposto anteriormente, outros tipos de algoritmo da família ACO também podem ser implementados, como é o caso do *MMAS*, do *Rank-based AS* e do *Population-based AS* (este último mais recente), que são estudados em diversos artigos e aplicações do TSP, problema semelhante ao das trajetórias.

Tendo em vista o grande tempo gasto pelos algoritmos na etapa de pré-processamento, podem-se estudar novos métodos de geração de malhas, talvez utilizar *softwares* já existentes na área de elementos finitos, visando gerar malhas contextualizadas e mais inteligentes para cada campo de velocidades. Essas malhas são enriquecidas em trechos de maior interesse da seção, como pontos onde existe um caminho preferencial das trajetórias do pulso ou onde existe um gradiente de

velocidades maior. O processo de criação mais inteligente gera malhas que se adequam melhor ao problema estudado e ao mesmo tempo apresentam menor quantidade de nós, diminuindo o tempo de montagem do grafo e de aplicação do algoritmo utilizado. Além disso, outras formas de montagem do problema podem ser adotadas na criação do grafo, como as listas de adjacência.

Pensando em aspectos computacionais, é possível escrever o código de aplicação dos algoritmos em linguagens de mais baixo nível, como é o caso do FORTRAN, já que as operações desempenhadas são relativamente simples e não há necessidade de bibliotecas elaboradas. Por ser de mais baixo nível, a linguagem executa as instruções mais rapidamente, gerando um código de melhor desempenho. Esse código pode ser exportado para o Pascal, linguagem na qual o *software* TUSom foi elaborado, na forma de DLL, uma extensão que executa apenas certos procedimentos (o algoritmo em questão), evitando a reprogramação de todo o *software* em outra linguagem. Além disso, podem ser aplicados paradigmas de programação paralela, como o OpenMP ou o MPI. Algoritmos como o ACS, em que as formigas caminham paralelamente, se adequam de forma natural a este tipo de programação, melhorando muito seu desempenho.

Refletindo sobre passos posteriores ao ajuste de trajetórias, pode-se simular medições experimentais após a correção das trajetórias e analisar o impacto do erro proveniente dos métodos aproximados na geração de imagens. Isso permitiria aumentar o nível de erro dos algoritmos bioinspirados até um limite que não prejudique tanto a qualidade das imagens, visando diminuir os tempos de execução destes algoritmos e melhorar seu desempenho.

Também é possível estudar a viabilidade de uso de alternativas inteligentes, como as redes neurais artificiais (RNAs), na geração de imagens. À partir de um mapa de velocidades, com a trajetória já ajustada, é possível gerar padrões de entrada experimental fictícios. Com essas entradas, pode-se treinar uma rede neural para que a mesma desempenhe a tarefa inversa, ou seja, à partir de entradas experimentais (agora reais, provindas de medições), gerar melhores mapas de velocidades.

O ajuste das trajetórias permite explorar o treinamento e aplicação futuros de redes neurais na geração de melhores imagens tomográficas à partir de medições experimentais, aprimorando a aplicação dos ensaios de ultrassom para monitoramento e detecção de heterogeneidades em estruturas de concreto.

Como dito anteriormente, as ideias são muitas e espera-se que os leitores possam fazer proveito de algumas dessas sugestões e descobrir novos caminhos, trazendo outras sugestões que retroalimentam o processo de investigação científica.

REFERÊNCIAS

AGGELIS, D. G. et al. Numerical simulation of elastic waves for visualization of defects. **Construction and Building Materials**, v. 25, n. 4, p. 1503-1512, 2011. DOI: 10.1016/j.conbuildmat.2010.08.008.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR 8802:** Concreto endurecido – Determinação da velocidade de propagação de onda ultrassônica. Rio de Janeiro: ABNT, 2019.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR 5738:** Concreto - Procedimento para moldagem e cura de corpos de prova. Rio de Janeiro, ABNT, 2016.

AZEVEDO, A. F. M.; **Método dos Elementos Finitos.** Porto: Faculdade de Engenharia da Universidade do Porto, 2003. Disponível em: http://www.alvaroazevedo.com/publications/books/livro_mef_aa_1ed/doc/livro_mef_aa.pdf. Acesso em: 11 jun. 2021.

CALVO, R. **Sistemas bio-inspirados para coordenação de múltiplos robôs móveis.** 2012. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2012. DOI: 10.11606/T.55.2012.tde-04092012-084821

CHAI, H. K. et al. Tomographic reconstruction for concrete using attenuation of ultrasound. **NDT & E International**, v. 44, n. 2, p. 206-215, 2011. DOI: 10.1016/j.ndteint.2010.11.003.

CHAI, H. K. et al. Singleside access tomography for evaluating interior defect of concrete. **Construction and Building Materials**, v. 24, n. 12, p. 2411-2418, 2010. DOI: 10.1016/j.conbuildmat.2010.03.003.

CIMMINO, G. Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. **La Ricerca Scientifica**, v.9, n. 2, p. 326-333, 1938.

CORMEN, T. H. et al. **Introduction to algorithms.** 3rd ed. Cambridge: The MIT Press, 2009.

DEANS, S. R. **The Radon transform and some of its applications**. New York: John Wiley & Sons, 1983.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische Mathematik**, v. 1, p. 269–271, 1959. DOI: 10.1007/BF01386390.

DORIGO, M.; BIRATTARI, M.; STÜTZLE, T. Ant colony optimization. **IEEE Computational Intelligence Magazine**, v. 1, n. 4, p. 28-39, 2006. DOI: 10.1109/MCI.2006.329691.

DORIGO, M.; STÜTZLE, T. **Ant colony optimization**. Cambrigde: The MIT Press, 2004.

DORIGO, M.; MANIEZZO. V.; COLORNI, A. Ant system: optimization by a colony of cooperating agents. **IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)**, v. 26, n. 1, p. 29-41, 1996. DOI: 10.1109/3477.484436.

DORIGO, M. **Optimization, learning and natural algorithms.** 1992. Thesis (PhD in System and Information Engineering) – Electronic Department, Politecnico di Milano, Italy, 1992.

ERTEL, W. **Introduction to artificial intelligence.** 2nd ed. Weingarten: Springer, 2017. (Undergraduate topics in computer science). DOI: https://doi.org/10.1007/978-3-319-58487-4

FERRARO, C. C.; BOYD, J. A.; CONSOLAZIO, R. G. Evaluation of damage to bridge piers using pulse velocity tomography. **Construction and Building Materials**, v. 38, p. 1303-1309, 2013. DOI: 10.1016/j.conbuildmat.2012.09.003.

FILLER, A. G. The history, development and impact of computed imaging in neurological diagnosis and neurosurgery: CT, MRI, and DTI. **Nature Proceedings**, v.7, n. 1, p. 1-85, 2009. DOI: 10.1038/npre.2009.3267.5.

GAMBARDELLA, L. M.; DORIGO, M. Solving symmetric and asymmetric TSPs by ant colonies. **Proceedings of IEEE International Conference on Evolutionary Computation**, p. 622–627, 1996. DOI: 10.1109/ICEC.1996.542672.

GENDREAU, M.; POTVIN, J. **Handbook of Metaheuristics.** 3rd ed. Cham: Springer, 2019. (International Series in Operations Research & Management Science). DOI: https://doi.org/10.1007/978-3-319-91086-4.

GIGLIO, V. M. HAACH, V. G. Uso da Otimização por Colônia de Formigas (ACO) na determinação da trajetória de pulsos ultrassônicos em elementos de concreto. *In:* XLI Ibero-Latin-American Congress on Computational Methods in Engineering (CILAMCE), 2020, Foz do Iguaçu. **Proceedings […]**, 2020. Disponível em: https://cilamce.com.br/arearestrita/apresentacoes/170/7688.pdf. Acesso em: 04 jul. 2021a.

GIGLIO, V. M. HAACH, V. G. Algoritmos genéticos na determinação da trajetória de ondas ultrassônicas para geração de imagens tomográficas em elementos de concreto. *In:* 62º Congresso Brasileiro do Concreto, 2020, Florianópolis. **Anais […]**, 2020. Disponível em: https://drive.google.com/file/d/1bMyBP0UXfDKlCa3rptCtzrUJX3Kte6CN. Acesso em: 04 jul. 2021b.

GOLDBERG, A. V.; HARRELSON, C. **Computing the shortest path:** A* search meets graph theory. Redmond: Microsoft Research, 2004. (Technical Report MSR-TR-2004-24). Disponível em: https://www.microsoft.com/en-us/research/wp-content/uploads/2004/07/tr-2004-24.pdf. Acesso em: 15 jun. 21.

GOSS, S. et al. Self-organized shortcuts in the Argentine ant. **Naturwissenschaften**, v. 76, p. 579–581, 1989. DOI: 10.1007/BF00462870.

HAACH, V. G. **Aplicação do ultrassom como ensaio não destrutivo para a avaliação de elementos de concreto.** 2017. Texto sistematizado (Livre docência) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

HAACH, V. G.; RAMIREZ, F. C. Qualitative assessment of concrete by ultrasound tomography. **Construction and Building Materials**, v. 119, p. 61-70, 2016. DOI: 10.1016/j.conbuildmat.2016.05.056.

HAACH, V. G.; JULIANI, L. M. Application of ultrasonic tomography to detection of damages in concrete. **Proceedings of the 9th International Conference on Structural Dynamics, EURODYN**, v. 9, p. 3351-3357, 2014. Disponível em: https://paginas.fe.up.pt/~eurodyn2014/CD/papers/469_MS22_ABS_1170.pdf. Acesso em: 13 jun. 2021.

HAUPT, R. L.; HAUPT, S. E. **Practical Genetic Algorithms**. 2nd ed. Hoboken: John Wiley & Sons, 2004.

HOLA, J.; SCHABOWICKZ, K. State-of-the-art non-destructive methods for diagnostic testing of building structures – anticipated development trends. **Archives of civil and mechanical engineering**, v. 10, n. 3, p. 5-18, 2010. DOI: 10.1016/S1644-9665(12)60133-2.

HOLLAND, J. H. **Adaptation in natural and artificial systems:** An introductory analysis with applications to biology, control and artificial intelligence. 2nd ed. Cambridge: The MIT Press, 1992.

JACKSON, M. J.; TWEETON, D. R. **MIGRATOM – Geophysical tomography using wavefront migration and fuzzy constraints.** U.S. Department of Interior – Bureau of Mines. Wasinghton, 1994.

KACZMARZ, S. Angenaherte auflosung von systemen linearer gleichungen, **Bulletin of the Polish Academy of Sciences: Technical Sciences**, v. 6-8A, p. 355-357, 1937.

KAK, A. C.; SLANEY, M. **Principles of Computerized Tomographic Imaging.** Philadelphia: Society for Industrial and Applied Mathematics, 2001. (Classics in Applied Mathematics).

KWON, J.; CHOI, S. J.; SONG, S. M. Implementation and evaluation of the ultrasonic TOF tomography for the NDT of concrete structures. *In:* ELECTRONIC IMAGING, 2005, San Jose. **Proceedings of SPIE 5674, Computational Imaging III**, 2005, 8p. DOI: 10.1117/12.588459.

LIMA, M. L. R. **Otimização topológica e paramétrica de vigas de concreto armado utilizando algoritmos genéticos.** 2011. Dissertação (Mestrado em Engenharia Civil (Estruturas)) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2011. DOI: 10.11606/D.3.2011.tde-17082011-153639.

LINDEN, R. **Algoritmos genéticos:** Uma importante ferramenta da inteligência computacional. Rio de Janeiro: Brasport, 2006.

MALHOTRA, V. M.; CARINO, N. J. **Nondestructive testing on concrete**. 2nd ed. Boca Raton: CRC Press, 2004.

MEHTA, P. K.; MONTEIRO, P. J. M. **Concreto:** Microestrutura, propriedades e materiais. 2. ed. São Paulo: IBRACON, 2014.

MICHALEWICZ, Z.; FOGEL, D. B. **How to solve it:** Modern heuristics. 2nd ed. New York: Springer-Verlag Berlin Heidelberg, 2004. DOI: 10.1007/978-3-662-07807-5.

MOSER, T. J. Shortest path calculation of seismic rays. **Geophysics**, v. 58, n. 1, p. 59-67, 1991. DOI: 10.1190/1.1442958.

OLIVEIRA et al. Analysis of the population-based ant colony optimization algorithm for the TSP and the QAP. *In:* 2017 IEEE Congress on Evolutionary Computation (CEC), 2017, Donostia/San Sebastián. **Proceedings […]**. Donostia/San Sebastián: IEEE, 2017. p. 1734-1741. DOI: 10.1109/CEC.2017.7969511.

PERLIN, L. P.; PINTO, R. C. A. Use of the network theory to improve the ultrasonic tomography in concrete. **Ultrasonics**, v. 96, p. 185-195, 2019. DOI: 10.1016/j.ultras.2019.01.007.

PERLIN, L. P. **Tomografia ultrassônica em concreto e madeira:** Desenvolvimento de ferramenta computacional. 2015. Tese (Doutorado em Engenharia Civil) – Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis, 2015. Disponível em: https://repositorio.ufsc.br/xmlui/handle/123456789/158870. Acesso em: 11 jun. 2021.

PERLIN, L.P.; PINTO, R.C.A. Tomografia ultrassônica em concreto. **Revista IBRACON de Estruturas e Materiais**, v. 6, n. 2, p. 246-269, 2013. DOI: 10.1590/S1983-41952013000200006.

PERLIN, L. P. **Tomografia ultrassônica em concreto:** Desenvolvimento de ferramenta computacional. 2011. Dissertação (Mestrado em Engenharia Civil) – Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis, 2011. Disponível em: https://repositorio.ufsc.br/xmlui/handle/123456789/95463. Acesso em: 11 jun. 2021.

PROCEQ SA. **Instruções operacionais: Pundit Lab/Pundit Lab^{+} - Instrumento Ultrassônico.** 2017. 32p. Disponível em: https://www.proceq.com/uploads/tx_proceqproductcms/import_data/files/Pundit%20Lab_Operating%20Instructions_Portuguese_high.pdf. Acesso em: 04 jul. 2021.

RAMÍREZ, F. C. **Detecção de danos em estruturas de concreto por meio de tomografia ultrassônica.** 2015. Dissertação (Mestrado em Engenharia Civil (Estruturas)) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2015. DOI: 10.11606/D.18.2015.tde-27052015-171423.

RIBEIRO, P. O. **Calibração automática de modelos em elementos finitos por meio de métodos de otimização e análise modal.** 2019. Dissertação (Mestrado em Engenharia Civil (Estruturas)) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019. DOI: 10.11606/D.18.2019.tde-26032019-111142.

ROMAN, N. T. **Estrutura de dados – Aula 23:** Grafos – Conceitos Básicos**.** 2017. Disponível em: <https://www.youtube.com/watch?v=MC0u4f334mI>. Acesso em: 06 out. 2019.

ROMAN, N. T. **Estrutura de dados – Aula 28:** Grafos – Algoritmo de Dijkstra**.** 2017. Disponível em: <https://www.youtube.com/watch?v=ovkITlgyJ2s>. Acesso em: 06 out. 2019.

RUSSEL, S. J.; NORVIG, P. **Artificial intelligence:** A modern approach. 3rd ed. Upper Saddle River: Prentice Hall, 2010 (Prentice Hall series in artificial intelligence).

SANTOS, R. E. dos. **A armação do concreto no Brasil: história da difusão da tecnologia do concreto armado e da construção de sua hegemonia.** 2008. 338 f. Tese (Doutorado em História da Educação), Faculdade de Educação, Universidade Federal de Minas Gerais, Belo Horizonte, 2008. Disponível em: https://repositorio.ufmg.br/handle/1843/FAEC-84KQ4X. Acesso em: 20 fev. 2021.

SANTOS, V. C. **Uma abordagem híbrida para planejamento exploratório de trajetórias e controle de navegação de robôs móveis autônomos.** 2017. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2017. DOI: 10.11606/T.55.2018.tde-07022018-144509.

SEDGEWICK, R.; WAYNE, K. **Algorithms, Part II.** 4th ed. Upper Saddle River: Addison-Wesley, 2014.

SOUZA, V. C. M. de; RIPPER, T. **Patologia, recuperação e reforço de estruturas de concreto**. São Paulo: Pini, 1998.

STÜTZLE, T.; HOOS, H. H. MAX–MIN Ant System. **Future Generation Computer Systems**, v. 16, n. 8, p. 889–914, 2000. DOI: 10.1016/S0167-739X(00)00043-1.

YANLI, C. The Shortest Path Ray Tracing Algorithm in Concrete Computerized Tomography. *In:* INTERNATIONAL FORUM ON INFORMATION TECHNOLOGY AND APPLICATIONS (IFITA), 2010, Kunming. **Proceedings [...]**. Kunming: IEEE, 2010. p. 392-396. DOI: 10.1109/IFITA.2010.307.

ZLOCHIN et al. Model-based search for combinatorial optimization: A critical survey. *In:* OPERATIONS RESEARCH 131, 2004, [S. l.]. **Annals [...]**. Netherlands: Kluwer Academic, 2004. p. 373-395. DOI: 10.1023/B:ANOR.0000039526.52305.af.