Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica

Algoritmo bioinspirado para a solução de roteamento de tráfego urbano considerando o conceito de Internet das Coisas

Autor:
Wesley Coelho Marciano

Orientador:
Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos
2016

Wesley Coelho Marciano

Algoritmo bioinspirado para a solução de roteamento de tráfego urbano considerando o conceito de Internet das Coisas

Trabalho de Coclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica – Ênfase em Eletrônica

ORIENTADOR: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos
2016

Marciano, Wesley Coelho
M319a Algoritmo bioinspirado para a solução de roteamento de tráfego urbano considerando o conceito de Internet das Coisas / Wesley Coelho Marciano; orientador Evandro Luis Linhari Rodrigues. São Carlos, 2016.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2016.

1. Engenharia Elétrica. 2. Mobilidade Urbana. 3. Sistema de Posicionamento Móvel. 4. Algoritmos Bioinspirados. 5. Internet das Coisas. 6. Redes. I. Título.

FOLHA DE APROVAÇÃO

Nome: Wesley Coelho Marciano

Título: "Algoritmo bioinspirado para a solução de roteamento de tráfego urbano considerando o conceito de Internet das Coisas"

Trabalho de Conclusão de Curso defendido e aprovado em 28/06/2016,

com NOTA 7,0 (Sete, zero), pela Comissão Julgadora:

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Dr. Ricardo Augusto Souza Fernandes - (DEE/UFSCar)

Mestre Michel Bessani - (Doutorando - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Dr. José Carlos de Melo Vieira Júnior

Agradecimentos

À Jesus e forças do bem, porque são fontes de energia positiva nos momentos mais críticos.

Aos meus pais, Rosemeire e Laucir, por abrirem mão de seus próprios sonhos e fazerem o possível e o impossível para oferecerem uma melhor oportunidade de vida a seus filhos.

Ao meu irmão, Willian, que sempre me ajudou quando precisei e que me incentivou no caminho dos estudos.

Aos meus parentes, em especial a meus tios Nucivaldo e Inês por todo o apoio e suporte.

A todos meus amigos e laços fortes que criei dentro da universidade, em especial à turma Elétrica 010.

Àqueles que tive o prazer de conhecer em projetos sociais que participei no decorrer da minha graduação, Inclusão, Projeto Semente e Campanha USP do Agasalho, pois me fizeram acreditar em um mundo melhor.

Aos meus professores orientadores de iniciação científica, Amílcar, e de projeto de formatura, Evandro, não só pela confiança em meu trabalho, mas também por suas dedicações à graduação.

À Escola de Engenharia de São Carlos e ao Departamento de Engenharia Elétrica, por propiciarem um leque de conhecimentos e ferramentas que me ajudaram na compreensão do mundo da tecnologia.

E à Universidade de São Paulo, pela oportunidade de crescimento pessoal e profissional em um ambiente de diversidade.

- Wesley Marciano

"A mente que se abre a uma nova ideia jamais voltará a seu tamanho original."

Albert Einstein

Resumo

A busca por soluções de roteamento nos novos tipos de arquiteturas de redes, baseados em comportamentos dinâmicos, heterogeneidade e larga escala, trazem a necessidade de algoritmos que considerem essas novas características, de forma a aumentar a eficiência e a inteligência das redes. Nesse sentido, bons resultados são conseguidos através do uso de algoritmos bioinspirados, provenientes do entendimento de mecanismos adquiridos pela Natureza no decorrer de sua evolução. Embora no estado-da-arte para solucionar problemas relacionados às telecomunicações, os algoritmos bioinspirados abrem uma gama de aplicações. O algoritmo de otimização por colônia de formigas (ACO), por exemplo, é muito eficiente na resolução de problemas onde há busca por um melhor caminho. Logo, também pode ser usado em situações semelhantes em que a finalidade seja a mesma: rápida convergência na busca por soluções de roteamento, como em um sistema de transporte dentro de cidades: o tráfego em centros urbanos é um ótimo cenário para a aplicação do ACO. Crescimento acelerado e não planejado das cidades geram problemas que afetam todos os setores da sociedades, entre eles, dificuldades de mobilidade e congestionamentos. Por outro lado, a evolução da tecnologia permitirá a internet das coisas, em que cada aparelho ou serviço estarão conectados entre si e dividindo informações. Assim, utilizando o ACO, este trabalho propõe um sistema de posicionamento móvel capaz de buscar e gerenciar rotas dentro de um espaço urbano do futuro, fazendo uso de uma rede totalmente acessível e das informações disponíveis nessa mesma rede e em que a internet das coisas já é uma realidade. Os resultados conseguidos neste trabalho indicam que o algoritmo proposto é uma possível direção para o roteamento de transportes urbanos e também a melhoria de seu desempenho mostra um caminho para contornar a limitação do ACO para dimensões elevadas.

Palavras-chave: Engenharia Elétrica, Mobilidade Urbana, Sistema de Posicionamento Móvel, Algoritmos Bioinspirados, Internet das Coisas, Redes.

Abstract

The search for routing solutions involving new types of network architectures based on dynamic behavior, heterogeneity and large scale asks for algorithms that consider these new characterisics, evolving the efficiency and intelligence of networks. In this way, good results are possible using bioinspired algorithms, originated from the understanding of mechanisms acquired in Nature on its evolution. Although in the state-of-the-art to solve problems that are related to telecommunications, bioinspired algorithms provides an array of applications. The ant colony optimization algorithm (ACO), for instance, is very effective in problems where search for best route is needed. Therefore this algorithm can also be used in similar situations where the goal is the same: fast convergence in the search for routing solutions, such as a transportation system in cities: traffic inside urban centers is a good scenario to apply ACO algorithm. Fast-growing and unplanned cities generate problems that affect all sectors of society, including mobility issues and traffic jam. On the other hand, the evolution of technology will allow internet of things, in which every single device or services will be connected and sharing information. Thus, using the ACO, this work propose a mobile positioning system that searches and manages routes inside a future city, making use a fully acessible network system and the informations available in this network, and where the internet of things is already a reality. The results obtained in this study indicate that the proposed algorithm is a possible direction for urban transport routing and also the improvement in its performance shows an alternative way to overcome the limitation of the ACO to higher dimensions.

Keywords: Electrical Engineering, Urban Mobility, Mobile Positioning System, Bioinspired Algorithms, Internet of Things, Networking.

Sumário

Introdução

Nas últimas décadas, o aumento da utilização das redes de comunicações tem causado problemas nos quais os algoritmos tradicionais não são mais eficientes [1]-[4]. Nesse sentido, a busca por soluções desses e outros problemas envolvendo os novos tipos de arquitetura de comunicações tem convergido para a utilização de algoritmos bioinspirados.

Há três áreas de pesquisas que envolvem a abordagem bioinspirada: 1) computação bioinspirada, em que o objetivo é aumentar a eficiência em computação; 2) sistemas bioinspirados, que se referem a arquiteturas distribuídas e colaborativas de sistema, como sensoriamento e exploração; e 3) redes bioinspiradas, que trata de estratégias para aumentar a eficiência de redes, principalmente de grande escala [1], [2].

Esses algoritmos resolvem problemas de otimização com base na emulação do comportamento da natureza. Destacam-se [1], [5] os algoritmos evolucionários (p. ex. algoritmos genéticos) e os de inteligência coletiva (sistemas imunológicos artificiais, inteligência de enxames de partículas, otimização por colônia de formigas). A **Figura 1** mostra a árvore que considera em quais áreas da Biologia o comportamento de cada um desses algoritmos é inspirado [5].

Algumas características dos algoritmos bioinspirados que interessam os pesquisadores na busca por analogias de aplicação e modelagem são capacidade de adaptação às variações ambientais, resiliência a falhas internas ou externas, capacidade de exibir comportamentos complexos através de um conjunto limitado

de regras básicas, capacidade de aprender e evoluir quando novas condições surgem, gerenciamento efetivo de recursos restritos com uma inteligência global maior do que a da superposição dos indivíduos, habilidade de se auto-organizar de forma totalmente distribuída e colaborativa alcançando o equilíbrio e capacidade de sobrevivência apesar de condições hostis do ambiente [1]-[3].

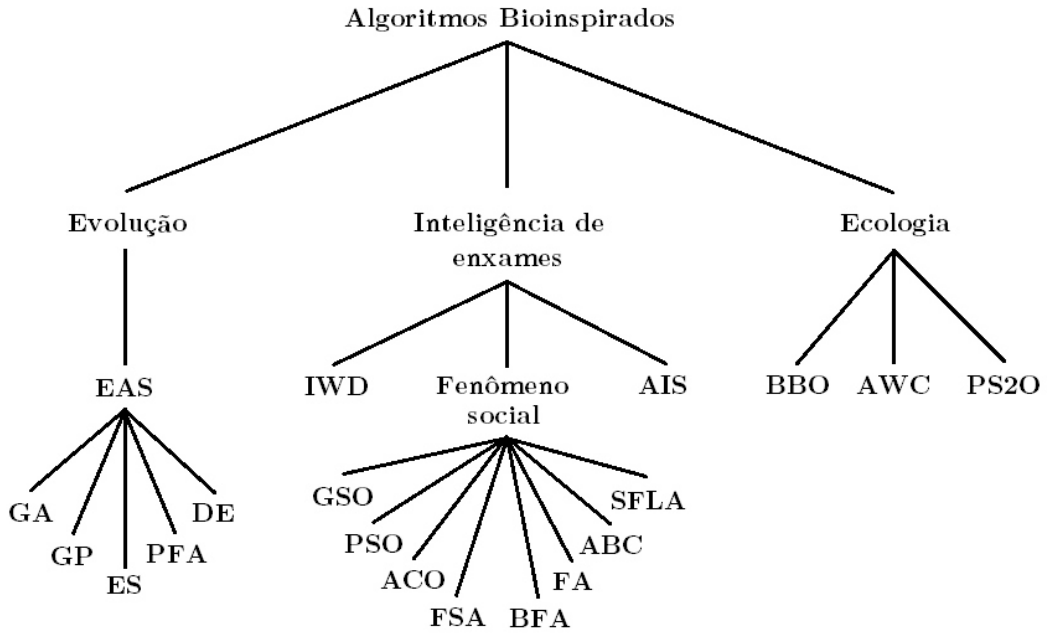

Figura 1: Algoritmos bioinspirados divididos por área de inspiração.

Embora esses algoritmos estejam em destaque especialmente para solucionar os problemas intrínsecos aos novos tipos de arquiteturas de redes, eles abrem um leque de aplicações que vão além da área de comunicações. Por exemplo, dentro dos algoritmos que descrevem inteligência coletiva [1], [5], [6], o algoritmo de otimização por colônia de formigas (ACO – *Ant Colony Optimization*), que é utilizado para solucionar problemas em que há necessidade de roteamento para dados em redes de larga escala, pode ser também aproveitado em qualquer outra aplicação cujo objetivo seja semelhante: rápida convergência de solução na busca por uma rota, dado um ponto inicial e um ponto final dentro de uma rede.

Exemplo de aplicação nesse sentido é um sistema de posicionamento móvel dentro de cidades, auxiliando no controle de tráfego urbano.

O crescimento acelerado e não planejado de cidades promovem problemas que afetam todos os setores da sociedade: segregação espacial, degradação do ambiente, falta de gerenciamento do espaço urbano, dificuldades de mobilidade, congestionamentos, entre outros [7], [8]. No caso dos problemas de mobilidade e congestionamento nas cidades, isso é um ótimo cenário para o estudo da aplicação de algoritmos bioinspirados na tentativa de melhorar o tráfego urbano, tanto a frota de veículos pessoais (p. ex. carros e motocicletas) quanto os transportes coletivos (p. ex. ônibus e metrô) [7].

Por outro lado, o avanço da tecnologia das comunicações permitirá a internet das coisas (IoT – *Internet of Things*), em que cada pessoa, aparelho ou serviço esterão conectados entre si e dividindo informações através de uma rede aberta e sem protocolos [9]-[13]. Algumas características da internet das coisas são: conectividade, inteligência, sensibilidade, expressividade, consumo eficiente, segurança, entre outras, que poderão propiciar o entendimento e gerenciamento do sistema de transporte urbano de forma dinâmica e em tempo real, por exemplo.

Nesse sentido, este trabalho visa a utilização de algoritmos bioinspirados, mais especificamente de um algoritmo baseado no ACO, aplicados para simular um sistema de posicionamento dentro de um espaço urbano do futuro, capaz de buscar e gerenciar rotas dentro da rede de transporte, dados os pontos de partida e de destino de cada móvel, levando em consideração os possíveis nós e conexões da rede e fazendo uso do conceito de conectividade da internet das coisas na obtenção e acesso a dados e informações.

Inicialmente são descritos os **Objetivos** do projeto, detalhando o problema estudado e os passos escolhidos para a sua solução. Em seguida, é apresentado o **Embasamento teórico**, mostrando as ferramentas que possibilitaram a simulação do sistema de posicionamento móvel. Depois são especificados os **Materiais e métodos** utilizados para o desenvolvimento do trabalho. A partir daí, os **Resultados e discussões** são apresentados. Após os resultados, a **Conclusão** do trabalho é exposta, assim como direções para possíveis melhorias e implementação do projeto. Por fim, os códigos elaborados para serem base de análise para este trabalho são mostrados no **Apêndice**.

Objetivos

No sentido de apontar um caminho para a solução do problema de mobilidade urbana em uma cidade do futuro, o trabalho feito é a simulação de um sistema de posicionamento móvel, o qual utilizou um algoritmo bioinspirado baseado no ACO para roteamento de móveis dentro de um espaço urbano. Também é importante ressaltar que essa proposta de solução considerou a dependência de uma rede baseada no conceito de internet das coisas, cujos pontos de recepção e transmissão de sinal fossem disposníveis a acesso, assim como a obtenção de dados e informações contidos nessa mesma rede.

Para alcançar esse objetivo, a análise deste trabalho foi dividida em três partes: estudo do ACO e sua modificação, testes de confiabilidade do algoritmo modificado e posicionamento móvel na rede.

Além disso, algumas hipóteses foram consideradas:

- O sistema/espaço urbano é aquele em que todos os meios de transporte são automatizados: velocidade e direção;
- A rede é homogênea, ou seja, todos pontos de acesso e enlaces possuem mesmo peso (não há prioridade);
- A internet das coisas é uma realidade. Todo aparelho, serviço e informação estão contidos na rede, por exemplo, localização do móvel e dos nós fixos da rede, distância entre esses nós, pontos de acesso e enlaces ativos ou inativos.

Embasamento teórico

Algoritmos e o ACO

Segundo Cormen, Leiserson, Rivest e Stein [14], algoritmo é qualquer procedimento computacional bem definido que, dado um conjunto de valores como entrada, produz algum valor ou conjunto de valores como saída. Entretanto, para conhecer completamente o procedimento computacional utilizado em um determinado problema, são necessários alguns conceitos básicos relacionados à complexidade computacional, otimização, qualidade da solução e técnicas de resolução [5], [6], [15], [16].

O primeiro refere-se ao custo de execução de um determinado procedimento/problema computacional, relacionando o tempo ou espaço de memória ao número de possíveis soluções. Assim, um problema computacional é polinomial determinístico (P) se o tempo de execução relaciona-se de forma polinomial com o número de soluções possíveis. Caso contrário, trata-se de um problema não polinomial determinístico (NP). Além disso, alguns problemas NP podem ser reduzidos a P, o que facilita a abordagem de resolução. No entanto, existem problemas em que não há como fazer essa simplificação e eles são chamados NP-completos (também conhecidos como NP-difíceis ou NP-*hard*).

Otimização consiste em minimizar ou maximizar uma função objetivo definida em um domínio. Nessa operação, pode-se obter uma vizinhança, isto é, conjunto de soluções aproximadamente iguais em um subconjunto do domínio, um mínimo/máximo local, se é a melhor solução dentro de uma vizinhança, e

ainda um mínimo/máximo global, se é a melhor solução dentro do domínio.

No que tange à qualidade da solução, define-se solução ótima aquela que apresenta o valor máximo ou mínimo global, isto é, apresenta a melhor solução para um determinado problema. Se não, define-se solução boa. A conveniência em se aceitar ou não este último tipo de solução depende do custo-benefício entre o tempo de execução do problema e sua proximidade com a solução ótima.

Por fim, as técnicas de resolução de problemas podem ser exatas (também chamadas ótimas), que garantem a obtenção da solução ótima, ou técnicas heurísticas, métodos aproximados que, embora não apresentem como resultado a solução ótima, garantem rapidez na busca por soluções boas.

Problemas com vários objetivos e/ou grande número de parâmetros fazem parte da classe de problemas não polinomial completos (NP-completos) cujas soluções mais adequadas pertencem à classe heurística. É para este tipo de solução que os algoritmos bioinspirados têm se tornado atraentes.

Em se tratando do algoritmo da otimização por colônia de formigas (ACO), ele pode ser aplicado na busca por rotas, característica importante em redes de larga escala e dinâmicas, como uma rede de transporte em espaços urbanos.

Formigas são excelentes exploradoras. São capazes de resolver tarefas complexas através de uma interação indireta entre os indivíduos dada por intermédio de modificações no meio ambiente, como o nível de feromônio deixado em rastros na busca por comida. O algoritmo é baseado nessa busca.

As formigas executam uma busca aleatória por comida e, se houver sucesso, a volta é marcada por um extenso rastro de feromônio entre o ninho e a localização da comida. Um número maior de indivíduos segue o caminho mais curto, aumentando ainda mais o nível de feromônio nesse rastro.

No movimento das formigas pela rede, o algoritmo tradicional do ACO [1], [2], [5], [6] leva em consideração a taxa atual de feromônio entre um nó e outro, além da visibilidade que uma formiga tem dos nós vizinhos a cada iteração, sendo pesos para o equacionamento.

Para a formulação do algoritmo, define-se a probabilidade de transição p_{ij} de uma formiga k se mover de uma posição i para uma posição j, dada por:

$$p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{l \in J_i^k} [\tau_{il}(t)]^{\alpha} \times [\eta_{il}]^{\beta}}, & se\ j \in J_i^k \\ 0, & caso\ contrário \end{cases}$$

na qual:

- J_i^k: são os nós ainda não visitados, isto é, uma formiga ao passar por algum desses nós evita visitar um nó i mais de uma vez;
- η_{ij}: é a visibilidade de j quando uma formiga está no nó i. Ou seja, o inverso da distância;
- τ_{ij}: é o nível de feromônio da aresta (i, j) ou a conveniência em se escolher o nó j estando em i;
- α e β: são parâmetros de ajuste que controlam respectivamente os pesos de nível de feromônio e visibilidade.

Após uma excursão, a formiga k deixa uma quantidade de feromônio $\Delta\tau_{ij}^{k}(t)$ em cada aresta (i, j). Essa variação de feromônio por uma formiga é dada por:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q / L^{k}(t), & se\ (i,j) \in T^{k}(t) \\ 0, & caso\ contrário \end{cases}$$

em que $T^{k}(t)$ é a viagem feita pela formiga k a cada iteração t, $L^{k}(t)$, a distância percorrida nessa viagem e Q, um parâmetro que controla o depósito de feromônio.

Outro fator importante que faz o algoritmo convergir para próximo de um ótimo é a busca aleatória que as formigas continuam a fazer mesmo depois de um sucesso em buscas anteriores. Com isso, a quantidade final de feromônio deixado em cada aresta e a cada iteração é $\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$, o que depende fortemente do número total m de formigas.

Redes sem fio e internet das coisas

Uma rede sem fio (ou *Wireless*) é um sistema que interliga vários equipamentos fixos ou móveis utilizando o ar como meio de transmissão. Proposta pelo IEEE (*Institute of Electrical and Electronics Engineers*) através do padrão 802.11, redes sem fio oferecem inúmeras vantagens frente a redes estáticas, dentre elas e mais importante, a mobilidade, uma vez que não necessitam estar ligadas a cabos [17], [18].

Segundo Gast [18], há quatro elementos básicos que precisam existir para

que se forme uma rede sem fio: estações (*stations*), dispositivos que recebem ou enviam dados; pontos de acesso (*access points*), que convertem protocolos em sinais; meio sem fio (*wireless medium*), que nada mais é do que ambiente que o sinal se propaga (p. ex. ar); e sistema de distribuição (*distribution system*), que conecta os vários pontos de acesso da rede. A **Figura 2** ilustra a disposição desses elementos.

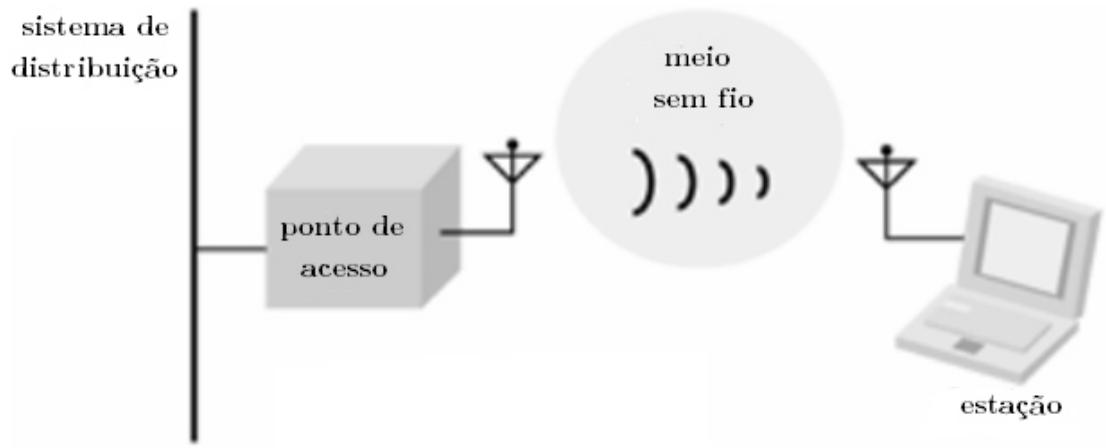

Figura 2: Componentes básicos de uma rede sem fio.

Os tipos de infraestrutura dessas redes, isto é, a disposição dos elementos dentro uma rede sem fio, além da sua escala, são variadas. Neste trabalho já se considera a internet das coisas e é nesse sentido que a proposta apresentada aqui se torna interessante, posto que a viabilidade de implementação do projeto é extremamente dependente da disponibilidade de acesso e de dados de localização dentro da rede, tanto os pontos de acesso quanto as estações (dispositivos) nos espaços urbanos, além de todas informações que podem estar atualizadas na rede, como tráfego de veículos e densidade de utilização de pontos de acesso.

Materiais e métodos

Para a elaboração dos códigos e simulação da proposta, foi utilizado o software MATLAB, desenvolvido e distribuído pela empresa MathWorks. O sistema operacional escolhido foi Microsoft Windows. As configurações da máquina utilizada para os testes foram Intel Core i5-5200U e 8GB de memória RAM.

Em relação ao algoritmo bioinspirado escolhido, utilizou-se uma versão simplificada do ACO original e também uma versão modificada do algoritmo a fim de comparação.

A ideia do ACO simplificado é mostrada abaixo, em que não se considerou os pesos de visibilidade e quantidade de feromônio: cada formiga é lançada na rede e, caso encontre um caminho que leva do ponto inicial ao final, uma quantidade de feromônio constante é depositada, independente da rota encontrada. Assim, o melhor caminho é aquele que apresenta a maior quantidade de feromônio: uma vez que o valor de feromônio é sempre o mesmo, independente do caminho encontrado, o nível dessa substância nos enlaces serão proporcionais ao inverso da distância do caminho determinado por cada formiga, ou seja, quanto mais curta a rota encontrada, maior a quantidade de feronômio a ser depositada nos enlaces que formam esse caminho.

Enquanto número m de formigas é maior que zero:
    Lançar formiga na rede;
    Se a formiga achar um caminho que leva do ponto inicial ao final:
        Depositar um número constante de feromônio nesse caminho;
    Caso contrário:

Não depositar feromônio;
Decrementar m;
Fim do laço.

No caso do ACO modificado proposto neste trabalho, levou-se em consideração apenas a distância total percorrida após o movimento de cada uma das formigas. A ideia do algoritmo modificado é explicada a seguir, em que o melhor caminho escolhido é aquele que apresenta a menor distância percorrida entre nó inicial e final, encontrado por apenas uma das formigas que passam pela rede.

Enquanto número m de formigas é maior que zero:
Lançar formiga na rede;
Se a formiga achar um caminho que leva do ponto inicial ao final:
Guardar caminho;
Se caminho for menor do que algum já encontrado:
Atualizar melhor caminho;
Decrementar m;
Fim do laço.

Seguindo a divisão proposta nos objetivos, a primeira parte foi voltada à análise do algoritmo bioinspirado. Para ilustrar a diferença entre o algoritmo original simplificado (ACO simplificado) e o modificado para este trabalho (ACO modificado), estudou-se uma rede quadrada de dimensão variável. Por exemplo, uma rede de dimensão 4 terá uma disposição de nós na forma 4x4 (total de 16 nós). A **Figura 3** mostra essa rede, em que foi atribuída uma distância de 100 unidades entre cada nó. Também foi definido que as formigas (e também os móveis) trafegam apenas horizontal ou verticalmente dentro da rede definida (linhas tracejadas da **Figura 3**). Escolheu-se redes quadradas/regulares devido à facilidade de cálculo de erro entre as rotas encontradas, além da conveniência

para o sistema de subredes proposto no ACO modificado, que será explicado posteriormente.

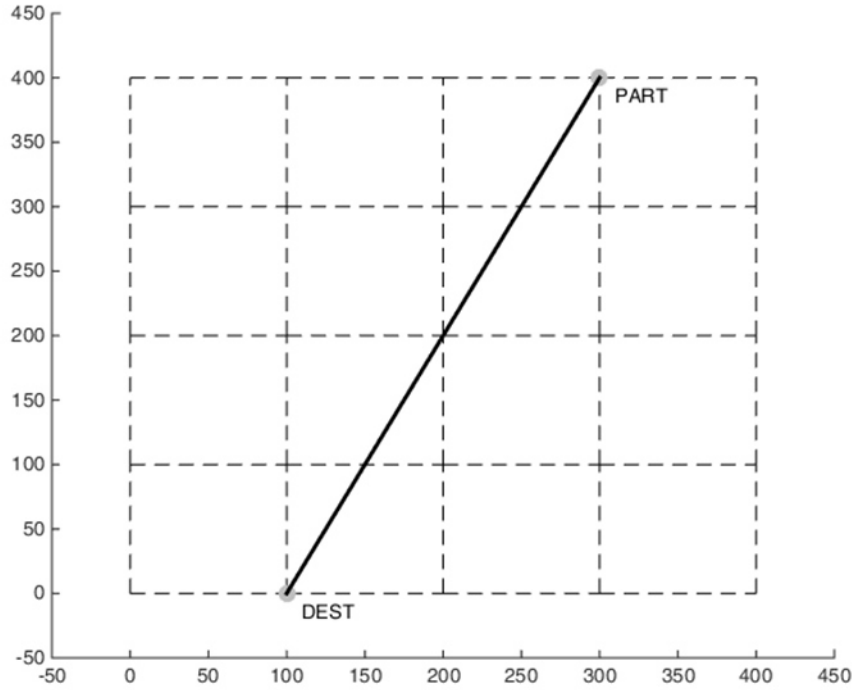

Figura 3: Exemplo de rede de dimensão 4 com distância cartesiana entre os pontos inicial e final em evidência.

A análise dos algoritmos foi baseada no erro entre a distância do caminho encontrado e a distância cartesiana entre os nós inicial e final (que é a menor possível). Foram utilizados 50 testes por dimensão de rede em cada um dos algoritmos, sorteando pontos iniciais e finais aleatórios a cada iteração. As dimensões analisadas variaram de 3 a 15. A **Figura 4** exemplica visualmente um dos testes do ACO modificado para uma rede de dimensão 11 com ponto de partida (900, 900) e de destino (0, 200), em que o erro é a relação percentual da diferença entre a distância total percorrida no caminho encontrado e a distância ideal (cartesiana entre os pontos de partida e destino). O número de formigas que passam pela rede a cada teste foi 500 em ambos os algoritmos.

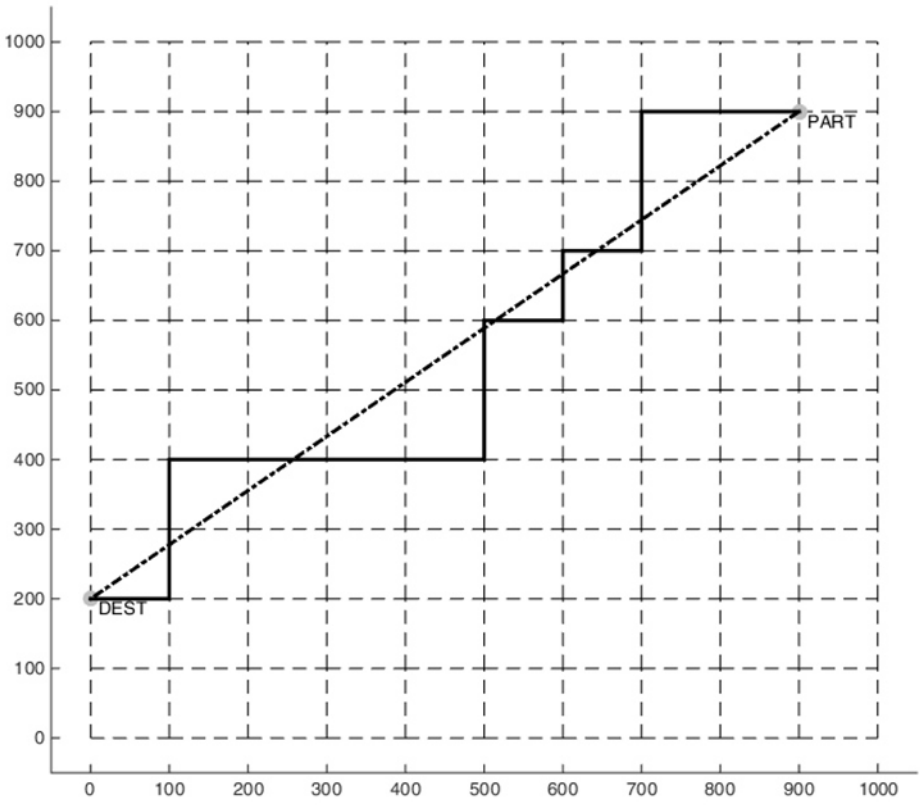

Figura 4: Exemplo de rede de dimensão 11 em que o caminho tracejado é a distância cartesiana entre pontos de partida e destino e o caminho com linha cheia é o encontrado pelo ACO modificado.

Após a demonstração da vantagem da utilização do ACO modificado, esse algoritmo foi analisado pelo ponto de vista do número de formigas que transitam pela rede, observando tanto o erro do caminho escolhido quanto o tempo de busca. Também se observou o tempo de busca ao variar a dimensão da rede.

Para a análise do erro e tempo de busca variando o número de formigas que transitam pela rede, adotou-se uma rede de dimensão 15 e número de testes de 50. Para estudo do tempo de busca variando-se a dimensão da rede, foram também adotados 50 testes e um número de formigas que transitam na malha

igual a 100.

Por fim, na última parte do projeto, analisou-se o movimento do móvel através da rede. É importante ressaltar que foi suposto que cada nó representa um ponto de acesso sem fio disponível na rede, como mostra a **Figura 5**, onde os pontos de acesso são WiFi, por exemplo.

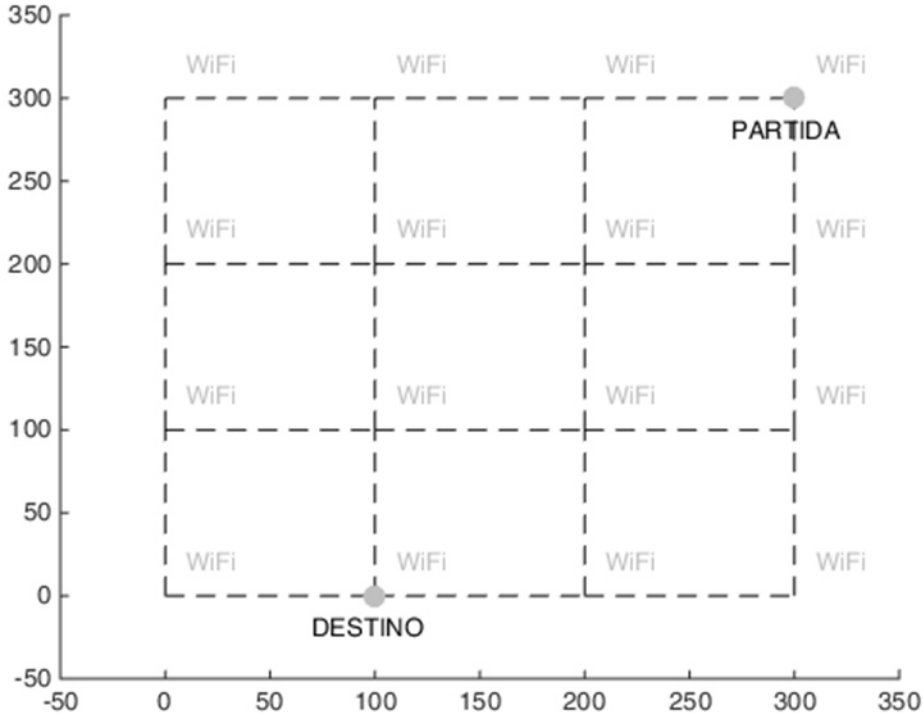

Figura 5: Exemplo de rede de dimensão 4 ilustrando que cada nó é um ponto de acesso disponível, por exemplo, um ponto WiFi.

Os programas desenvolvidos encontram-se disponíveis no **Apêndice**. No entanto, para a execução completa dos códigos, é necessário um pequeno trecho que define uma função de distância cartesiana ente dois pontos. Esse trecho pode ser criado como uma função em MATLAB e é mostrado a seguir.

```
function [ modulo ] = distancia(x1, y1, x2, y2)

% Esta função calcula a distância entre dois vetores.

        modulo = sqrt((x1-x2)^2 + (y1-y2)^2);

end
```

Resultados e discussões

Primeiramente, para a aferição dos códigos implementados, o ACO original foi testado e comparado a resultados teóricos. Depois disso, ele foi adaptado para as condições do ACO simplificado e do ACO modificado.

Ainda antes de comentar os resultados obtidos da comparação entre os algoritmos, é interessante ressaltar o funcionamento do ACO, apesar do código comentado apresentado no **Apêndice**. Inicialmente é determinado um número de formigas que serão introduzidas na rede. No caso do ACO simplificado, cada formiga busca um caminho que a leva do ponto de partida ao ponto de destino e, se encontrado algum caminho, uma quantidade constante de feromônio por formiga é despejada nessa rota. Caso a formiga não encontre uma rota, não há deposição de feromônio. No caso do ACO modificado, a diferença é que a distância total percorrida pelo caminho encontrado por cada formiga é computado. Se outra formiga encontra um caminho cuja distância seja menor, então esse passa a ser o melhor caminho encontrado. Assim, no algoritmo modificado, a menor distância é encontrada por uma das formigas, adotada como a melhor solução.

Tanto no ACO simplificado quanto no ACO modificado, são necessários alguns dados e parâmetros importantes para o algoritmo: uma matriz contendo as coordenadas de cada um dos nós que compõem a rede, uma matriz de enlace que indica se há ou não conexão entre o nó da linha i com o nó da linha j (restrição), um vetor de nós já visitados por cada formiga para que ela não volte por um mesmo ponto (restrição), dimensão da subrede no caso do ACO modificado (restrição) e uma matriz de feromônio que indica a quantidade dessa

substância em cada enlace (apenas no caso do ACO smplificado), além dos valores de dimensão e número de formigas que são lançadas na rede.

O comportamento das formigas para buscar um caminho é semelhante em ambos os casos. Para demonstrar esse comportamento do algoritmo, é adotada a rede e pontos de partida e destino da **Figura 6**.

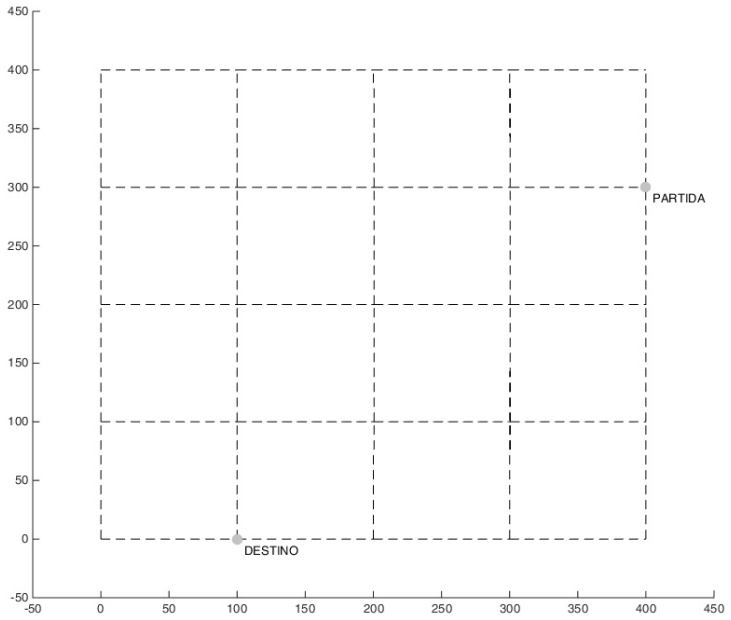

Figura 6: Rede criada para estudo do comportamento das formigas.

Em um primeiro momento, a formiga detecta que há três possíveis arestas para serem seguidas saindo do ponto de partida adotado (**Figura 7a**) e escolhe aleatoriamente uma delas. Além disso, o algoritmo não permite que a formiga volte a um mesmo nó visitado e, por isso, há três novas possibilidades de escolha na **Figura 7b**. Seguindo a mesma ideia anterior, a situação da **Figura 7c** indica apenas duas arestas para serem escolhidas. Caso a formiga selecione o caminho da direita (ainda se referindo à **Figura 7c**), há então apenas um único segmento a ser seguido: o que leva ao nó do ponto de partida (**Figura 7d**). Posto que não é possível voltar a um nó já visitado anteriormente, então significa que o algoritmo

não encontrou uma rota, logo não será depositado feromônio ou computada a distância nessa situação ou outra parecida. Caso a formiga ache um caminho que leve ao ponto de destino, aí sim há deposição de feromônio ou é computada a distânica da rota encontrada.

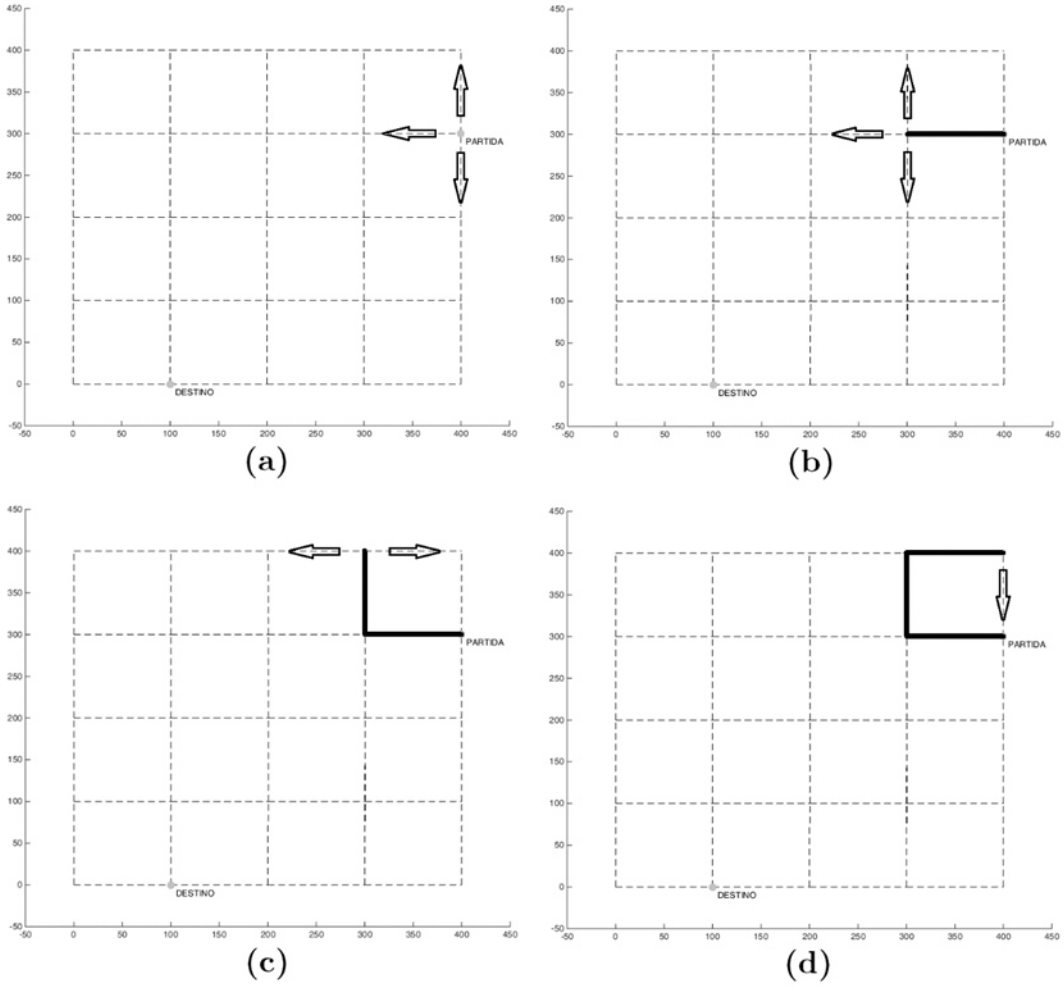

Figura 7: Exemplificação de uma formiga que não encontra uma rota que liga os pontos de partida e destino.

No caso do ACO simplificado, há várias rotas encontradas pelas formigas que transitam pela rede. No entanto, apenas o caminho que apresenta a maior deposição de feromônio será o escolhido pelo algoritmo como solução. Por outro lado, o ACO modificado computa apenas o caminho encontrado por uma das formigas: o que apresenta menor distância. Os resultados da comparação entre os

algoritmos do ACO simplificado (baseado no nível de feromônio das arestas) e do modificado seguem nos gráficos das **Figuras 8 e 9**.

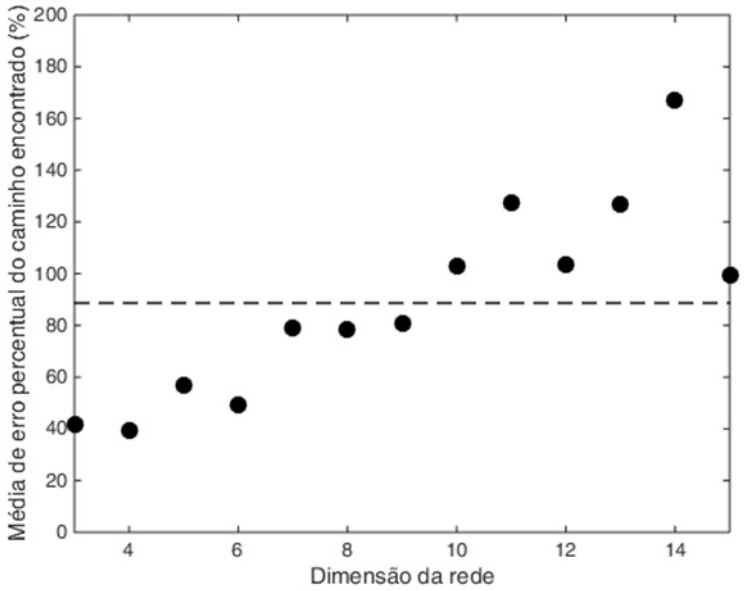

Figura 8: Resultados da análise do ACO simplificado relacionando a dimensão da rede ao erro do caminho encontrado. A linha tracejada (média) tem valor 88,64%.

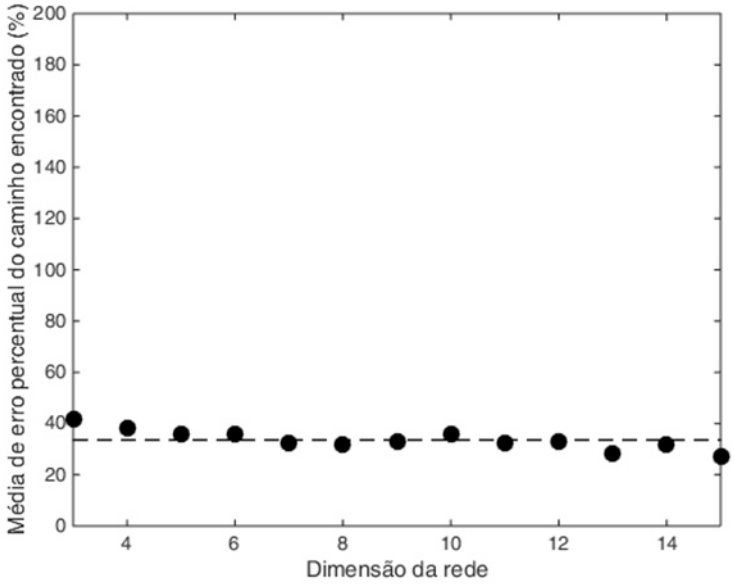

Figura 9: Resultados da análise do ACO modificado relacionando a dimensão da rede ao erro do caminho encontrado. A linha tracejada (média) tem valor 33,58%.

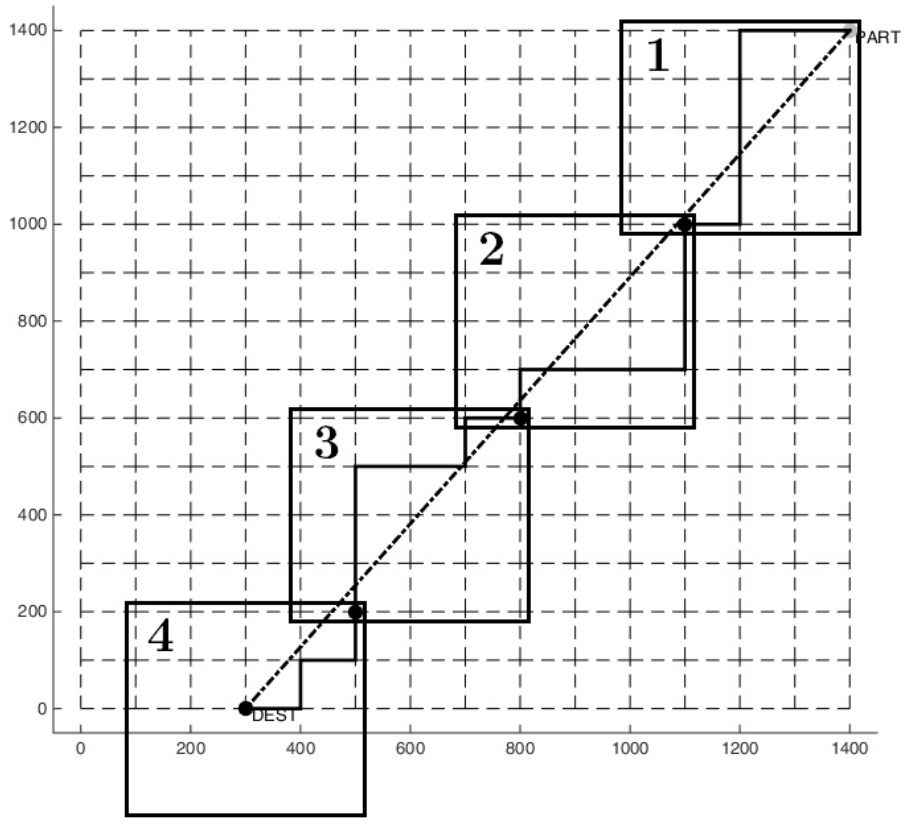

Figura 10: Figura que ilustra o comportamento do ACO modificado, em que a busca é dividida por subredes. Os pontos intermediários em evidência indicam os nós finais de cada subrede.

Os resultados indicam que a aleatoriedade do ACO simplificado se torna um problema para redes de dimensão grande (**Figura 8**), ao passo que o erro do ACO modificado se mantém em torno de uma constante (**Figura 9**). A vantagem do algoritmo modificado é que a busca do melhor caminho é feita através de subredes, o que otimiza o processo de busca por controlar a aleatoriedade do processo de busca do algoritmo, como ilustra a **Figura 10**, em que 4 subredes de dimensão 5 são criadas até o ponto de destino. O ponto final dentro de cada subrede é baseado no caminho ideal, isto é, na reta que indica a distância cartesiana entre os pontos de partida e destino. A **Figura 11** indica alguns caminhos encontrados pelo ACO simplificado. É interessante notar algumas soluções inesperadas ou indesejadas, como a primeira imagem.

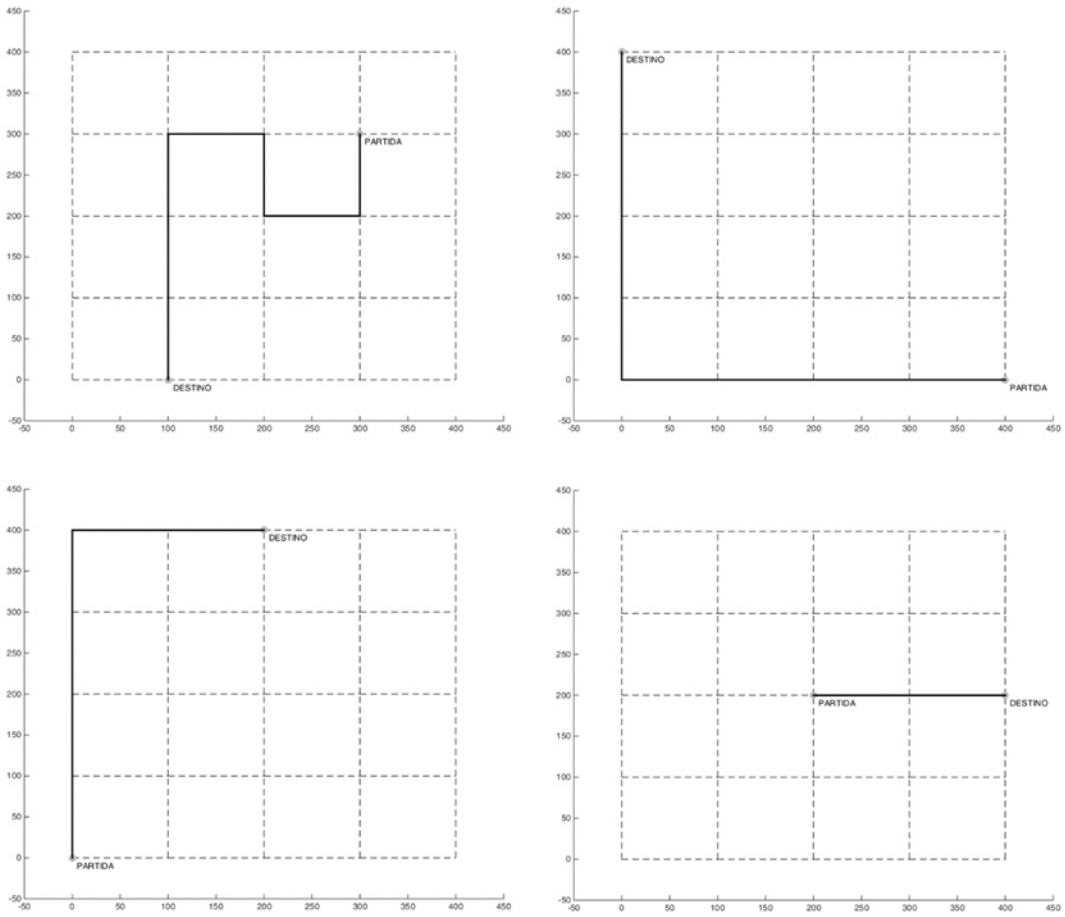

Figura 11: Resultado da aplicação do ACO simplificado para busca de rota variando os pontos de partida e de destino do móvel.

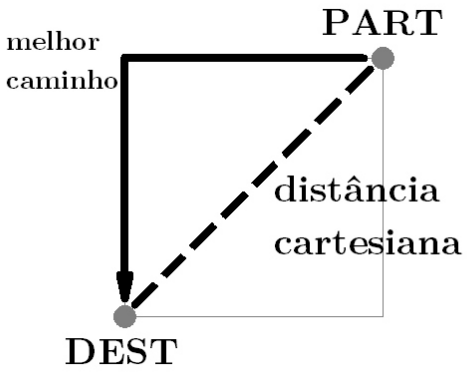

Figura 12: Menor entidade da rede, isto é, dimensão 2x2. O tracejado indica a distância cartesiana e a flecha o melhor caminho possível de ser encontrado.

Uma importante análise a se fazer é sobre o erro encontrado no ACO modificado entre o melhor caminho escolhido e a distância ideal entre pontos de partida e destino, que teve um média de 33,58% (**Figura 8**). Ao observar a menor entidade da rede, ela seria como a da **Figura 12**.

Como se trata de uma rede quadrada, há um triângulo retângulo formado pela linha tracejada e o melhor caminho. Sendo assim, adotando d o comprimento da aresta:

$$distância\ ideal = \sqrt{d^2 + d^2} \quad \text{e}$$

$$distância\ flecha = 2d$$

$$\text{então, } \frac{distância\ flecha}{distância\ ideal} - 1 = 0{,}4142$$

Ou seja, um erro próximo de 41,42% entre um caminho encontrado e a distância cartesiana entre os pontos inicial e final é o máximo no caso de uma rede quadrada. A média de erro de 33,58% no caso do ACO modificado em relção aos 88,64% do ACO simplificado mostra um melhor desempenho do algoritmo após a modificação.

Ilustrada a vantagem do ACO modificado, outros testes foram feitos para estudar a confiança do algoritmo. Entre eles, foi analisado se o número de formigas que transitam na rede influenciam no erro do caminho encontrado. A **Figura 13** mostra de antemão que o erro foi praticamente inerte frente à variação do número de formigas que passam pela rede: mesmo para um número de formigas próximo a 40 ou 50 o algoritmo manteve a taxa de erro observado em valores mais elevados desse parâmetro, como 900 ou 1000.

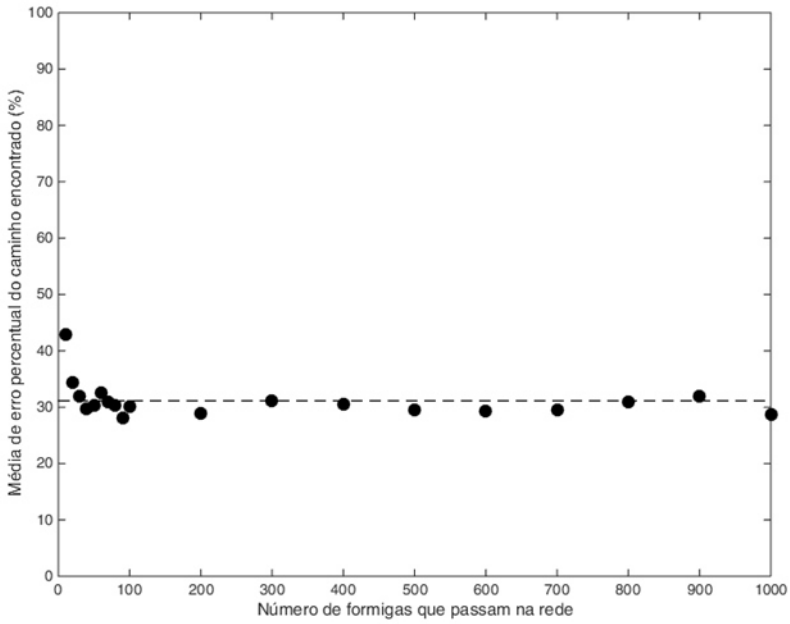

Figura 13: Resultados da análise do ACO modificado relacionando o número de formigas que transitam na rede ao erro do caminho encontrado. A linha tracejada (média) tem valor 31,14%.

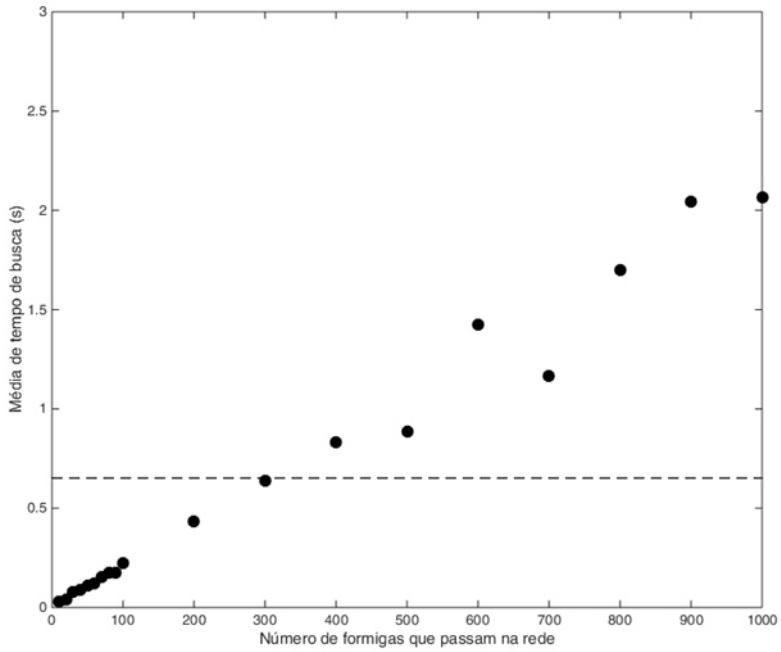

Figura 14: Resultados da análise do ACO modificado relacionando o número de formigas que transitam na rede à média de tempo de busca por uma solução. A linha tracejada (média) tem valor 0,65 segundos.

Outro teste feito foi a análise do tempo de busca da melhor solução variando-se também o número de formigas. A **Figura 14** mostra os resultados obtidos.

À medida que se aumentou o número de formigas lançadas na rede, o tempo de solução também foi incrementado. Isso é esperado, uma vez que a busca feita por cada formiga envolve um determinado esforço computacional. Assim, quanto maior esse número de formigas, maior o esforço computacional e, consequentemente, o tempo de busca será crescente. No entanto, aliando esse resultado ao da **Figura 13**, o ACO modificado se torna mais viável ao se usar um menor número de formigas, como 200, 100 ou até 50, posto que o erro é praticamente invariável.

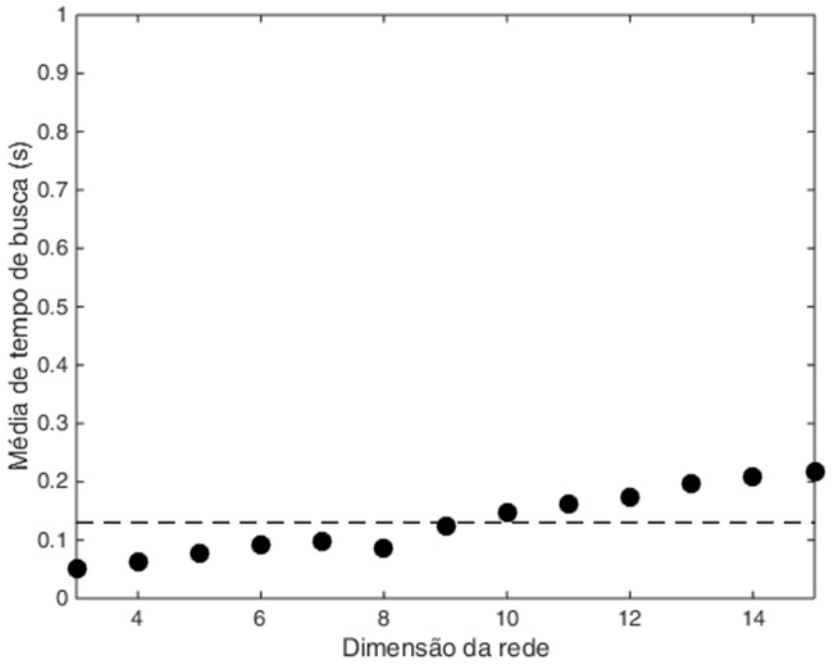

Figura 15: Resultados da análise do ACO modificado relacionando a dimensão da rede à média de tempo de busca por uma solução. A linha tracejada (média) tem valor 0,13 segundos.

O último teste feito para a analisar o desempenho do algoritmo foi o comportamento do tempo de busca ao se variar a dimensão da rede. Isso é ilustrado no gráfico da **Figura 15**, que mostra um ligeiro incremento à medida em que se aumenta a dimensão da rede, explicado pela soma do tempo de busca em cada subrede.

Para a última parte do algoritmo, em que o móvel é direcionado através das arestas do melhor caminho definido pelo ACO modificado, foi proposto a ativação de três pontos de acesso por vez. Isso foi pensado da seguinte forma: para cada segmento/enlace foi considerado seus pontos inicial e final. Em cada um desses pontos (que já são definidos dentro da rede), foram procurados pontos de acesso em seus arredores, incluindo os próprios nós em questão, cujas distâncias fossem menor que 200 (por se tratar de uma rede quadrada com arestas medindo 100 unidades). Assumindo novamente a rede da **Figura 6** (rede de dimensão 5 com ponto de partida de abcissa 400 e ordenada 300), a primeira aresta pela qual o móvel passa poderia ser a mostrada na **Figura 16**. O ponto de partida seria um dos nós da extremidade do enlace; o nó ao final do primeiro segmento, o outro. A **Figura 17**, por sua vez, ilustra os pontos de acesso disponíveis ao redor de cada um dos nós de extremidade da aresta. Também foi considerado que todos os pontos de acesso ao redor dos pontos e enlaces de interesse estão disponíveis/ativos durante todo o percurso pelo caminho encontrado.

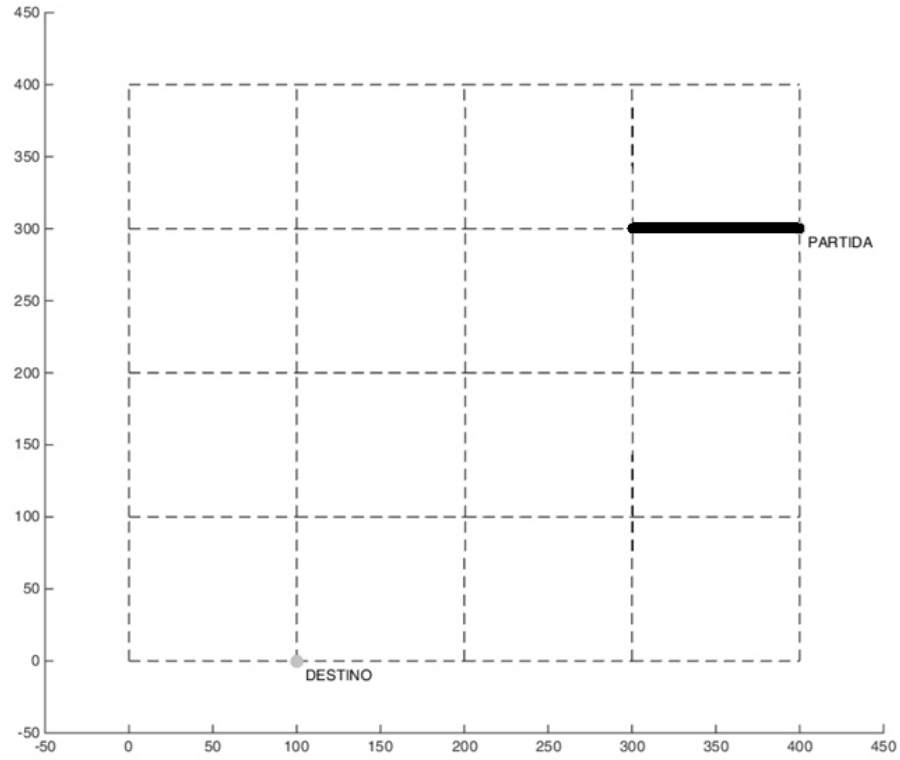

Figura 16: Primeiro segmento do melhor caminho a ser percorrido pelo móvel.

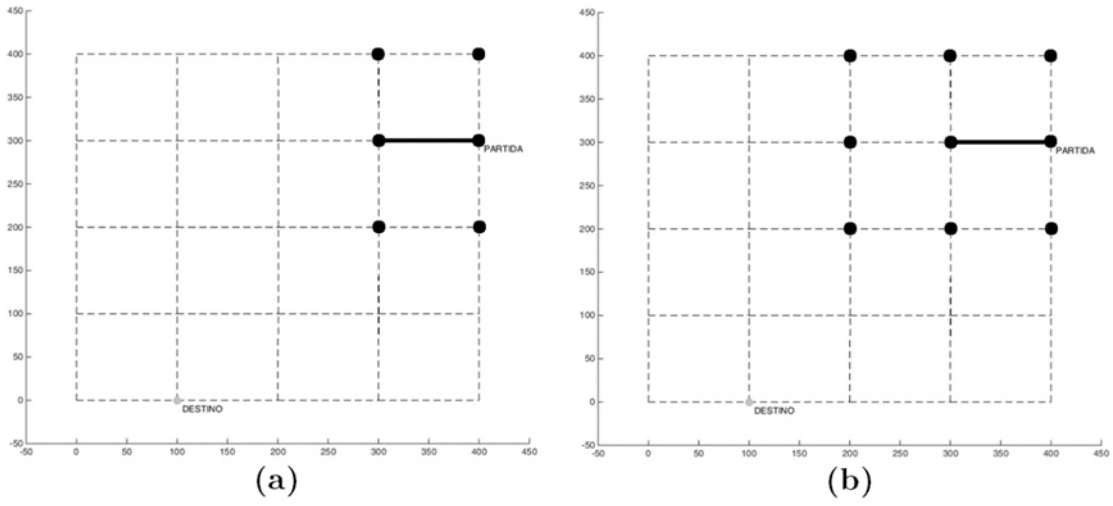

Figura 17: Pontos de acesso disponíveis ao redor (a) do ponto de partida e (b) da outra extremidade do segmento atual.

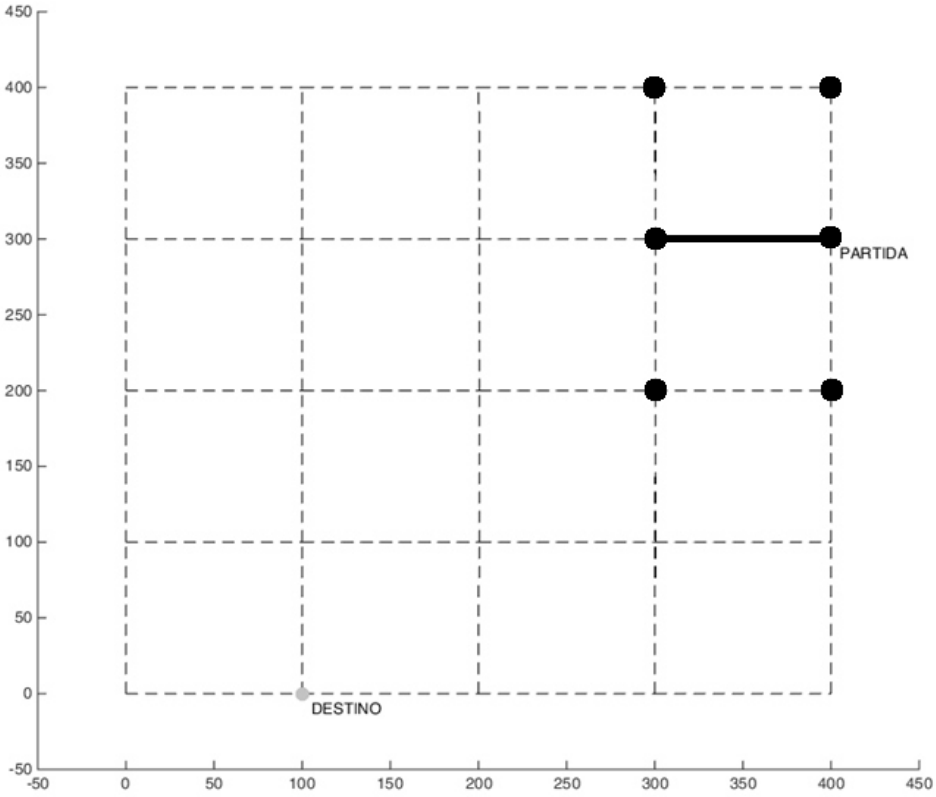

Figura 18: Pontos de acesso mais interessantes para o segmento em questão. Dentre esses pontos, três serão escolhidos aleatoriamente e ativados.

Supondo o alcance de redes sem fio limitadoo ideal é que os três pontos de acesso a serem ativados estejam o mais próximo possível de cada segmento que o móvel percorre. Também não é de interesse ter pontos de acesso ativados longe do móvel, uma vez que isso é importante para o entendimento e controle do tráfego local. Dessa forma, os pontos de acesso mais interessantes são conseguidos através da interseção entre aqueles das **Figuras 17a e 17b**. A **Figura 18** mostra tal resultado e, consequentemente, os três pontos de acesso escolhidos estarão nessa interseção. E esse processo se repete a cada segmento do melhor caminho escolhido pelo ACO modificado até que o móvel atinja seu destino.

A escolha de três pontos de acesso a cada aresta se deve ao fato de que é

necessário, no mínimo, três pontos para que se forme uma base e que se consiga ter um posicionamento exato do móvel na rede, considerando um sistema de coordenadas cartesianas.

O resultado do algoritmo completo é exemplificado na **Figura 19**, que mostra a disposição de escolha dos pontos de acesso ao longo de um caminho escolhido através do ACO e percorrido por um móvel.

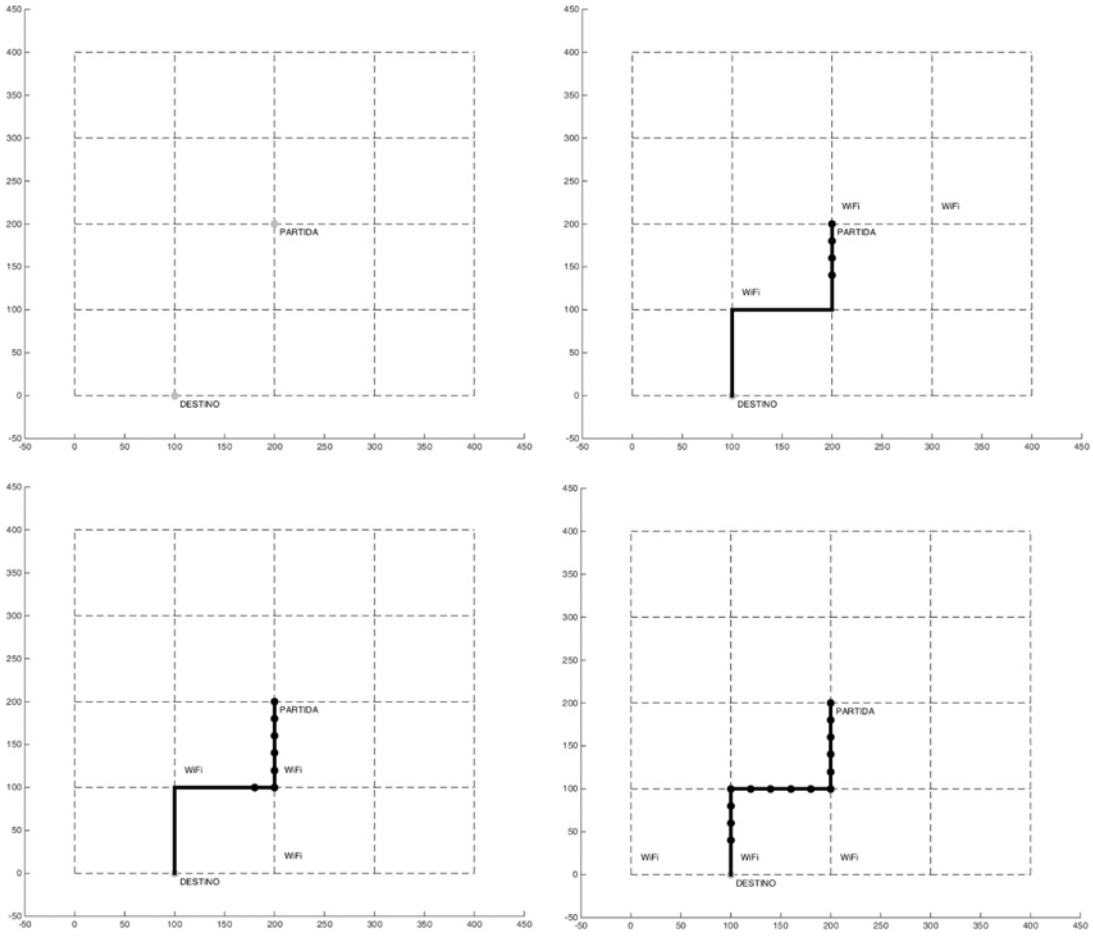

Figura 19: Pontos de acesso, p.ex. WiFi, ativados ao longo do caminho percorrido por um móvel.

Um ponto importante é que alguns problemas podem surgir quando muitos

móveis transitam pela rede, tais como congestionamento das arestas e dos pontos de acesso. A **Figura 20** apresenta algumas soluções do ACO modificado quando uma das as arestas é inativada através de um valor de threshold pré-determinado (por exemplo, um número máximo de móveis por aresta). O mesmo tipo de situação pode ser feita para um ponto de acesso fique sobrecarregado: nesse caso, o nó ou ponto deve se tornar inativo.

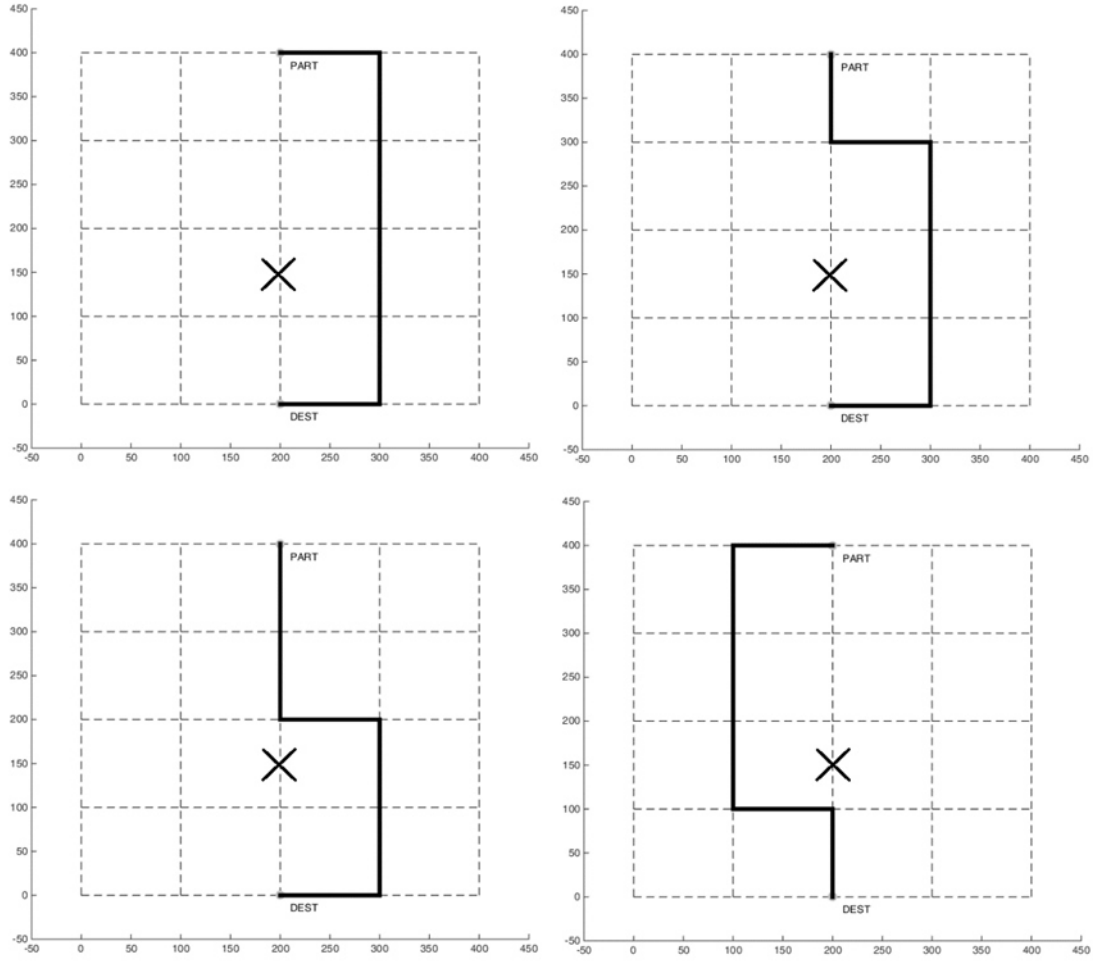

Figura 20: Comportamento do ACO modificado quando um dos enlaces se torna inativo por congestionamento. Sem a condição de threshold, o caminho seria uma reta entre os nós inicial e final.

Uma última análise a ser feita é sobre o ACO modificado frente a outros

algoritmos de roteamento tradicionais, como Dijkstra e Fuzzy, avaliando o tempo de busca por soluções, que variam de 0 a 0,1 segundos no caso de Dijkstra [19], de 0 a 1,0 segundos no caso de Fuzzy [20] e lembrando que a média de tempo encontrada neste trabalho para o ACO modificado é de 0,13 segundos. O sistema operacional, ambiente de simulação e configurações da máquina utilizados não são informados nas referências tomadas como base, mas os resultados servem para demonstrar a melhoria de desempenho do ACO modificado frente ao original quando da observação do tempo de busca de outros algoritmos de roteamento.

Conclusão

Os algoritmos bioinspirados chamam a atenção quando no uso em problemas pertencentes à classe heurística e que envolvem grande número de parâmetros. No caso do problema de busca por rotas, típico problema em que a aplicação de algoritmos bioinspirados é conveniente, o principal resultado é que esse tipo de abordagem garante uma rápida convergência para uma solução boa, o que é desejado em redes de grande escala e tempo real [1]-[6], como é o caso de uma rede de transporte. É observado também que essa solução boa depende dos parâmetros adotados, tais como dimensão da rede/subrede e número de formigas adotado.

No caso específico do ACO modificado, ele se mostrou eficiente e eficaz na busca por rotas em redes de dimensões variadas, sendo uma possível alternativa ou complemento aos algoritmos tradicionais, p.ex. Dijkstra ou Fuzzy, posto a proximidade dos tempos de busca mesmo em ambientes e sistemas diferentes de simulação.

Ainda sobre a análise do ACO, é comum na literatura a análise da função fitness do algortimo, relacionando a convergência da solução ao número de iterações (ou formigas). Essa análise não foi feita neste trabalho devido à modificação dos algoritmos, tanto no ACO simplificado quanto no ACO modificado: a proposta adotada foi que cada formiga lançada na rede não sofreria influência da modificação deixada pelas formigas anteriores. Ou seja, não há uma convergência para um caminho com maior feromônio durante o incremento das iterações no ACO simplificado: é apenas computada a quantidade de feromônio

deixada por cada formiga e depois é escolhida a rota em que a quantidade dessa substância seja maior. No caso do ACO modificado, apenas o caminho percorrido por cada formiga é computado e analisado.

Um outro erro possível de ser analisado acerca do ACO modificado é em relação à distância ortogonal do caminho escolhido em relação à distância ideal (**Figura 21**). No entanto, apenas o comprimento da distância foi analisado devido ao tipo de aplicação, em que o combustível e gasto de energia, que variam de acordo com a distância percorrida, são fatores mais importantes a serem considerados em meios de transporte.

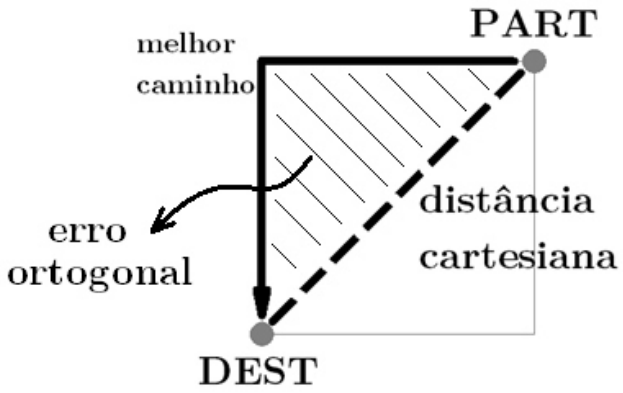

Figura 21: Esquema que exemplifica o erro ortogonal: a distância modular ao longo do melhor caminho em relação à reta que contém a distância cartesiana.

Sobre o tipo regular de rede escolhida para o trabalho, adotou-se redes quadradas a fim de facilitar a análise do erro da distância encontrada pelo algoritmo e também pela praticidade de se implementar o sistema de subredes. Todavia, ele pode ser adaptado para uso em mapas reais, como mostra a **Figura 22**, em que cada cruzamento pode ser tratado como um nó da rede.

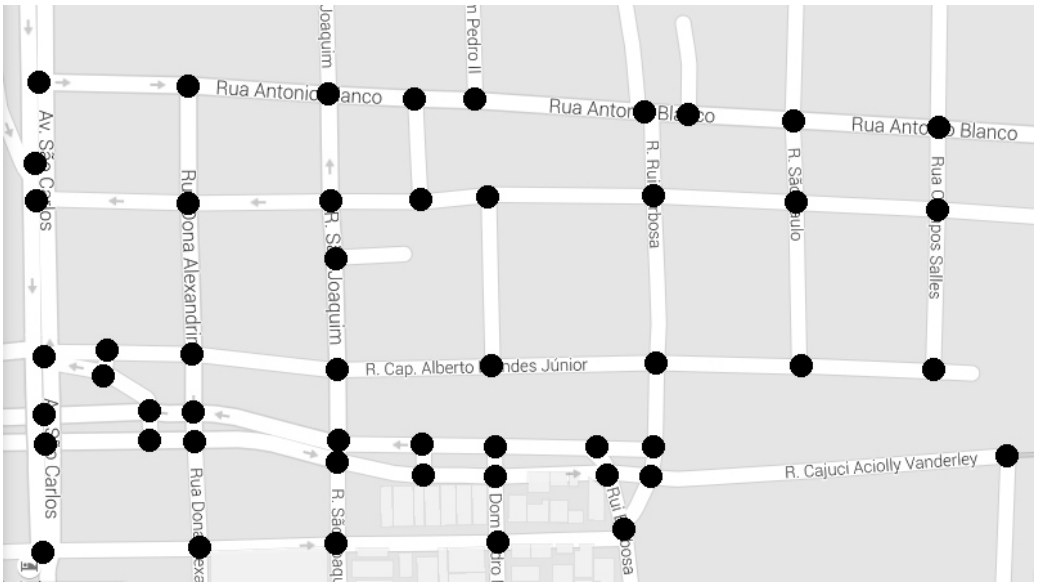

Figura 22: Trecho de mapa da cidade de São Carlos retirada do Google Maps, em que os nós são colocados em evidência visando a aplicação do ACO modificado.

Em relação ao controle de congestionamentos e mobilidade, neste trabalho foi considerado que um enlace ou ponto de acesso é inativado apenas quando uma condição de threshold é satisfeita, influenciando o comportamento dos móveis da rede apenas a partir daquele momento. Entretanto, também se pode aplicar um sistema de multiagentes, em que o melhor caminho encontrado por cada um dos móveis também seja atualizado para outros meios de transporte, influenciando também no comportamento global de roteamento quando há um possível grande fluxo da frota para uma determinada região da rede.

Trazendo a proposta desse trabalho para o mundo real, alguns problemas podem surgir de antemão quando da sua implementação:

- Segurança e anomalias da rede;
- Mapeamento dos pontos de acesso;
- Precisão do posicionamento dos móveis.

Qualquer sistema, seja de pequena ou grande escala, está sujeito a erros de operação ou ataques cibernéticos. Ainda mais quando se trata da internet das coisas, em que todos serviços e pessoas estão conectados simultaneamente e dividindo informações: uma rede dessa escala e aberta é suscetível a ataques e mal funcionamentos a todo momento. Uma possível saída para esse problema se encontra nos próprios algoritmos bioinspirados, como o Sistema Imunológico Artificial, também dentro do grupo de algoritmos de inteligência coletiva [1], [5], [21].

No caso dos pontos de acesso, é fundamental que todos estejam mapeados na rede. No caso de uma rede ser aberta, isto é, a qualquer momento usuários poderem habilitar ou não suas redes particulares para acesso geral, é necessário uma forma eficiente de detectar e incluir esses novos pontos no sistema. Tendo como hipótese a facilidade de acesso a fibras ópticas em um futuro próximo, uma possível solução seria utilização dos algoritmos RTT (ou *Round-Trip Time*), que utilizam o envio/recebimento de sinais de um ponto a outro para determinar a distância entre eles [22]. É importante ressaltar que a determinação exata das coordenadas de cada ponto de acesso é essencial para que não haja acúmulo de erro, posto que essas coordenadas são utilizadas para posicionamento do móvel e de entradas e saídas para outras informações da rede.

A precisão da posição dos móveis (ou estações) dentro da rede também se torna um problema dependendo do grau de exatidão que se pretende. Nessa direção, vários algoritmos e propostas vem sendo estudadas, como algoritmos de triangulação (**Figura 23**), estimação face a face, RFID, entre outros [23]-[27].

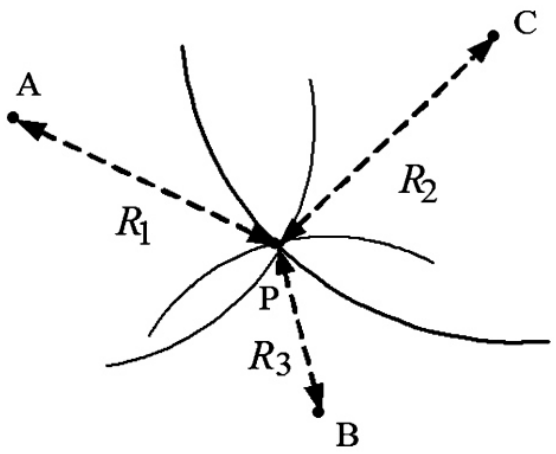

Figura 23: Esquemático de determinação da posição do móvel através de tempos de envio e recebimento de um sinal a outros pontos [24].

Uma comparação inevitável que surge é entre o projeto proposto nesse trabalho e um GPS por satélite. Diferente do posicionamento por satélite, cujos pontos de acesso (os próprios satélites) são geoestacionários, ou seja, têm uma posição bem definida para servirem como base de coordenadas para a definição de longitude e latitude dos móveis [28], as bases do ACO modificado são dinâmicas e dependentes dos pontos de acesso disponíveis da rede. Sendo assim, há necessidade de algoritmos eficientes e rápidos de mudança de base, cujas soluções podem ser encontradas na geometria analítica ou álgebra linear, p. ex. mudança de bases em R^2 [29], [30].

Em relação à qualidade do sistema, também é interessante citar a importância do tempo de resposta da rede. Em uma situação em que todos os transportes dentro de um espaço urbano são autônomos, o intervalo entre o recebimento de dados de posição pode significar um deslocamento considerável do móvel, causando desvio de sua rota, por exemplo.

Enfim, utilizando e otimizando algumas soluções propostas para os problemas de implementação ou utilizando algoritmos híbridos, p.ex. entre os três algoritmos de roteamente citados neste trabalho, o projeto se mostra uma possível direção para a solução do gerenciamento da mobilidade nas cidades na medida em que as redes crescem cada vez mais, assim como seu acesso dentro dos espaços urbanos. Além disso, a proposta apresentada de modificação do algoritmo (divisão em subredes) se mostra como um possível caminho no sentido de superar a limitação do ACO em para redes de grande dimensão.

Referências

[1] Dressler, F., Akan O.B., 'A Survey of bio-inspired networking', Computer Networks, 54 (2010), pp. 881-900.

[2] Dressler, F., Akan O.B., 'Bio-Inspired Networking: From Theory to Practice', IEEE Communications Magazine, November 2010.

[3] Lio, P., Verma, D., editores convidados, IEEE Networking, v.24, n.3, Maio/Junho 2010. Edição especial sobre redes bioinspiradas.

[4] Gao, S., editor, 'Bio-Inspired Computacional Algorithms and Their Applications', Intech, ISBN 978-953-51-0214-4, disponível em www.intecgopen.com/books, 2012.

[5] Binitha, S., Siva Sathya, S., 'A Survey of Bio-inspired Optimization Algorithms', International Journal of Soft Computing and Engineering, 2012.

[6] Serapião, Adriane Beatriz de Souza, 'Fundamentos de Otimização por Inteligência de Enxames: Uma Visão Geral', Revista Controle & Automação/Vol.20 no.3/Julho, Agosto e Setembro 2009.

[7] Costa, M. da S., da Silva, A.N.R., 'Um índice de mobilidade urbana sustentável', São Carlos, 2008.

[8] Resende, P.T.V. de, Sousa, P.R. de, 'Mobilidade urbana nas grandes cidades brasileiras: um estudo sobre os impactos do congestionamento', Caderno de Ideias C/0910, Fundação Dom Cabral, 2009.

[9] Petera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D., 'Context Aware Computing for The Internet of Things: A Survey', IEEE Communications Surveys & Tutorials, Vol.16, No.1, 2014.

[10] Atzori, L., Iera, A., Morabito, G., 'The Internet of Things: A survey',

Computer Networks 54 (2010) 2787-2805.

[11] Borgia, E., 'The Internet of Things vision: Key features, applications and open issues', Computer Communications 54 (2014) 1-31.

[12] Miorandi, D., Sicari, S., Pellegrini, F. De, Chlamtac, I., 'Internet of Things: Vision, applications and research challenges', Ad Hoc Networks 10 (2012) 1497-1516.

[13] Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M., 'Internet of Things (IoT): A vision, architectural elements, and future directions', Future Generation Computer Systems 29 (2013) 1645-1660.

[14] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 'Algoritmos: Teoria e Prática', 2ª Ed., 2002, Elsevier Editora Ltda.

[15] Iglesias, M.S., Machado, V.P., de Paula, P.H.O., Ribeiro, M.R.N., 'Introdução aos algoritmos bioinspirados nos cursos de graduação e sua aplicabilidade em problemas de alta complexidade computacional', artigo publicado no XXXVIII Congresso Brasileiro de Educação em Engenharia, COBENGE 2010.

[16] França, F.O. de, 'Algoritmos Bioinspirados Aplicados à Otimização Dinâmica', Dissertação de Mestrado, Universidade Estadual de Campinas, 2005.

[17] ANSI/IEEE Std 802.11, 1999 Edition.

[18] Gast, M., '802.11 Wireless Networks: The Definitive Guide', O'Reilly, Second Edition, 2005.

[19] Dramski, M., 'A comparison between Dijkstra algorithm and simplified ant colony optimization in navigation', Scientific Journal, Maritime University of Szczecin, 2012, 29(101), pp.25-29.

[20] Arnold, W., Hellendoorn, H., Seising, R., Thomas, C., Weitzel, A.,

‘Fuzzy Routing’, Fuzzy Sets and Systems 85 (1997) 131-153.

[21] Tarek S. Sobha, Wael M. Mostafa, 'A cooperative immunological approach for detecting network anomaly', 2010 Elsevier B.V.

[22] Tsang, Y., Yildiz, M., Barford, P., Nowak, R., ‘Network Radar: Tomography from Round Trip Time Measurements’.

[23] Shala, U., Rodriguez, A., ‘Indoor Positioning using Sensor-fusion in Android Devices’, 2011.

[24] Banerjee, P., Liu, J., ‘Survey of Wireless Indoor Positioning Technoques and Systems’, 2007.

[25] Higuchi, T., Yamaguchi, H., Higashino, T., ‘Context-supported local crowd mapping via collaborative sensing mobile phones’, 2013.

[26] Liu, S., Jiang, Y., Striegel, A., ‘Face-to-Face Proximity Estimation Using Bluetooth On Smartphones’, 2014.

[27] Welbourne, E., Battle, L., Cole, G., Gould, K., Rector, K., Raymer, S., Balazinska, M., Borriello, G., ‘Building the Internet of Things Using RFID’, IEEE Internet Computing 33 (2009) 48–55.

[28] Parthasarathy, J., ‘Positioning and Navigation System Using GPS’, 2006.

[29] Boulos, P., Camargo, I. de, ‘Geometria Analítica, Um Tratamento Vetorial’, 2ª Edição.

[30] Zani, S.L., ‘Álgebra Linear’, Departamento de Matemática, ICMC-USP, 2007.

Apêndice

A – Posicionamento móvel via rede WiFi utilizando o ACO original

```
% SISTEMA DE POSICIONAMENTO MÓVEL ACO:

% Apagando todos os dados e fechando janelas:
clear all
close all
clc

%% PARTE 1 - DEFINIÇÃO DA REDE E DOS PONTOS DE PARTIDA E DESTINO

% Declaração das variáveis globais:
global DIM;     % Dimensão da rede (DIMxDIM);
global PART;    % Ponto de partida do carro.
global DEST;    % Ponto de destino do carro.
global ITER;    % Número de iterações (formigas que passam na rede) do ACO.

% Definição das variáveis globais:
DIM = 5;
ITER = 500;

% Variáveis globais que serão definidas pelo usuário:
PART = [300; 400];
DEST = [100; 0];

% Definição dos pontos:
% Cada ponto é fisicamente um nó da rede e vão de 1 a DIMM^2.
cont = 1;
pontos = zeros(2,DIM^2);
for j=1:DIM
    for i=1:DIM
        pontos(1,cont) = (i-1)*100;
        pontos(2,cont) = (j-1)*100;
        cont=cont+1;
    end
end

% Matriz conexão:
% Essa matriz define se um ponto da linha i tem conexão com o ponto da
% da coluna j. É definido 1 se há e 0 se não há conexão.
matriz_conexao = zeros(DIM^2);

for i=1:DIM^2
    if i==1
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i+DIM)=1;
    elseif i>1 && i<DIM
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i+DIM)=1;
    elseif i==DIM
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i+DIM)=1;
    elseif i>1 && mod((i-1),DIM)==0 && i<(DIM^2-DIM+1)
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-DIM)=1;
        matriz_conexao(i,i+DIM)=1;
    elseif i>DIM && mod(i,DIM)==0 && i<DIM^2
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i-DIM)=1;
        matriz_conexao(i,i+DIM)=1;
    elseif i==(DIM^2-DIM+1)
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-DIM)=1;
    elseif i>(DIM^2-DIM+1) && i<DIM^2
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-DIM)=1;
    elseif i==DIM^2
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i-DIM)=1;
    else
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-DIM)=1;
```

```
        matriz_conexao(i,i+DIM)=1;
    end
end

% Imprimindo a rede:
figure('units','normalized','outerposition',[0 0 .5 .8])
for i=1:size(matriz_conexao,1)
    for j=i:size(matriz_conexao,2)
        if matriz_conexao(i,j)==1
            line([pontos(1,i) pontos(1,j)],[pontos(2,i) pontos(2,j)],...
                 'Color','k','LineStyle','--','LineWidth',0.1);
            hold;
        end
    end
end
% Eixos do gráfico:
% x: -50 a 350.
% y: -50 a 250.
axis([-50 50+(DIM-1)*100 -50 50+(DIM-1)*100]);

% Imprimindo os pontos WiFi na rede:
strwf = 'WiFi';
for i=1:size(pontos,2)
    text(pontos(1,i)+10,pontos(2,i)+20,strwf,'Color','w','BackgroundColor','w');
end

% Encontrando o ponto inicial escolhido para o carro:
for a=1:size(pontos,2)
    if PART(1)==pontos(1,a) && PART(2)==pontos(2,a)
        no_inicial = a;
        break;
    end
end

% Encontrando o ponto final escolhido para o carro:
for a=1:size(pontos,2)
    if DEST(1)==pontos(1,a) && DEST(2)==pontos(2,a)
        no_final = a;
        break;
    end
end

% Imprimindo o ponto de partida na figura:
text(pontos(1,no_inicial)+10,pontos(2,no_inicial)-15,'PART');
hold on;
plot(pontos(1,no_inicial),pontos(2,no_inicial),...
'Marker','.','Color',[.75 .75 .75],'MarkerSize',30);

% Imprimindo o ponto de destino na figura:
text(pontos(1,no_final)+10,pontos(2,no_final)-15,'DEST');
hold on;
plot(pontos(1,no_final),pontos(2,no_final),...
'Marker','.','Color',[.75 .75 .75],'MarkerSize',30);

%% PARTE 2 - APLICAÇÃO DO ACO PARA BUSCAR UM CAMINHO

% Criando uma matriz de distância entre pontos. É uma matriz tipo
% simétrica.
m_distancia = zeros(size(pontos,2));
for i=1:size(pontos,2)
    for j=i:size(pontos,2)
        m_distancia(i,j) = distancia(pontos(1,i),pontos(2,i),pontos(1,j),pontos(2,j));
        m_distancia(j,i) = m_distancia(i,j);
    end
end

% Inicializando uma matriz de feromônio:
m_feromonio = zeros(size(pontos,2));

% Busca do melhor caminho utilizando o ACO:
no_atual = size(pontos,2)+1;
while no_atual~=no_final
    for i=1:1:ITER

        % Inicializando flag para saída de loop:
        sair = 0;

        % Inicializando um contador auxiliar:
        contador = 0;

        % Inicializando um vetor que indica quais nós ainda foram ou não
        % visitados. Se nó da coluna j já foi visitado, então recebe valor 0.
        % Caso contrário, recebe 1.
        v_visitados = ones(1,size(pontos,2));

        % Inicializando um vetor auxiliar para indicar o caminho percorrido por
```

```
% cada formiga.
v_caminho = zeros(1,size(pontos,2));

% O primeiro valor do caminho a ser percorrido é o próprio ponto de
% partida:
v_caminho(1,1) = no_inicial;

% O nó atual indica onde cada formiga se encontra a cada movimento na
% rede:
no_atual = no_inicial;

% Enquanto flag de saída não for acionada:
while sair==0

    % Se o nó atual for diferente do nó final, então continua
    % procurando um caminho:
    if no_atual~=no_final || sair~=1

        % Inicializando um vetor linha, cujo conteúdo indica os
        % caminhos, i.e, pontos possíveis de seguir a cada nó:
        v_saidas = zeros(1,4);

        % Contador é incrementado (inicializado com zero):
        contador = contador+1;

        % O primeiro nó do caminho a ser percorrido, i.e, o nó inicial
        % é um nó visitado:
        v_visitados(v_caminho(contador))=0;

        % Inicializando uma variável que mostra o número total de
        % possíveis saídas do nó atual:
        saidas = 0;

        % Verificando as possíveis saídas para o nó atual:
        for j=1:size(pontos,2)

            % Verificando quantas e quais saídas são as possíveis para
            % o nó  atual. Se uma das possíveis saídas é um nó já
            % visitado, essa possibilidade não é contabilizada através
            % da multiplicação pelo vetor de nós visitados:
            if matriz_conexao(v_caminho(contador),j)*v_visitados(j)==1
                saidas = saidas + 1;
                v_saidas(saidas) = j;
            end
        end

        % Sorteando uma saída aleatoriamente para cada caso:
        dado = 10*rand;
        switch saidas

            % Se não houver mais saídas, então não há mais caminho a
            % pecorrer
            case 0
                sair = 1;
            % Caso o número de saídas seja um, então há apenas um
            % caminho a seguir.
            case 1
                v_caminho(contador+1) = v_saidas(1);
            % Caso houver duas possíveis saídas, a variável dado é
            % acionada e, se for menor que 5, a primeira saída do vetor
            % linha de saídas é selecionada. Caso contrário, a segunda
            % saída possível é selecionada.
            case 2
                dado = randperm(2,1);
                if dado==1
                    v_caminho(contador+1) = v_saidas(1);
                else
                    v_caminho(contador+1) = v_saidas(2);
                end
            % Caso o número de saídas possíveis sejam três, um sorteio
            % é feito semelhante ao caso anterior, porém com as
            % possibilidades divididas em três.
            case 3
                dado = randperm(3,1);
                if dado==1
                    v_caminho(contador+1) = v_saidas(1);
                elseif dado==2
                    v_caminho(contador+1) = v_saidas(2);
                else
                    v_caminho(contador+1) = v_saidas(3);
                end
            % Caso o número de saídas possíveis sejam quatro, um sorteio
            % é feito com as possibilidades divididas em quatro.
            case 4
                dado = randperm(4,1);
                if dado==1
```

```
                        v_caminho(contador+1) = v_saidas(1);
                    elseif dado==2
                        v_caminho(contador+1) = v_saidas(2);
                    elseif dado==3
                        v_caminho(contador+1) = v_saidas(3);
                    else
                        v_caminho(contador+1) = v_saidas(4);
                    end
                otherwise
            end % Fim do switch com variável saída.

            % Atualizando nó atual:
            if v_caminho(contador+1)~=0
                no_atual = v_caminho(contador+1);
            end

            % Checando se o nó atual é o final:
            if no_atual==no_final
                sair = 1;
            end

        end % Fim da condição if com variável flag.

    end % Fim do laço para encontrar ou não um caminho.

    % Se foi encontrado um caminho cujos nós inicial e final sejam os
    % desejados, então há depósito de feromônio nesse caminho. A quantidade
    % total de feromônio depositada é sempre a mesma para qualquer caminho.
    % Sendo assim, quanto menor esse caminho, maior quantidade de feromônio
    % será depositada nele.

    distancia_total = 0;
    if no_atual==no_final
        % Encontrando distância total do caminho percorrido, caso seja de
        % interesse:
        for j=2:size(v_caminho,2)
            if v_caminho(j)~=0
                distancia_total = distancia_total + ...
                distancia(pontos(1,v_caminho(j-1)),pontos(2,v_caminho(j-1)),...
                pontos(1,v_caminho(j)),pontos(2,v_caminho(j)));
            end
        end

        % Atualizando nível de feromônio na conexões:
        for j=2:size(v_caminho,2)
            if v_caminho(j)~=0
                m_feromonio(v_caminho(j-1),v_caminho(j)) = ...
                m_feromonio(v_caminho(j-1),v_caminho(j))+(1/distancia_total);
                m_feromonio(v_caminho(j),v_caminho(j-1)) = ...
                m_feromonio(v_caminho(j-1),v_caminho(j));
            end
        end

    end % Fim do if que compara se nó atual equivale ao nó final.

end % Final da iteração do número de formigas.

% Testando se a matriz de feromônio contém elementos não nulos. Caso sim,
% foi encontrado um caminho. Caso não, nenhum caminho foi encontrado:

% Inicializando variável que informa se há ou não um caminho encontrado:
ha_caminho = 0;

% Percorrendo matriz de feromônio:
for i=1:(size(pontos,2)^2)
    if m_feromonio(i)~=0
        ha_caminho = 1;
        break;
    end
end

% Após o número de formigas desejado que passaram na rede,
% verificando qual é o caminho que apresenta maior quantidade de
% feromônio, caso exista. Esse caminho será o escolhido como ótimo.

if ha_caminho==1 && no_atual==no_final

    % Criando um vetor que indicará o melhor camiho a ser escolhido:
    melhor_caminho = zeros(1,size(pontos,2));

    % Utilizando a mesma ideia de nó atual:
    no_atual = no_inicial;

    % Inicializando flag de saída de loop:
    sair = 0;
```

```
        % Iniciando loop que encontrará o melhor caminho:
        for i=1:size(melhor_caminho,2)

            % Atualizando melhor caminho:
            melhor_caminho(i) = no_atual;

            % Se condição de saída for acionada, sai do loop.
            if sair==1
                break;
            end

            % Variável auxiliar que indica maior valor de feromônio a cada linha:
            maior = 0.0;

            % Verificando qual o maior valor de feromônio na linha do nó atual da
            % matriz de feromônio e salvando novos valores de feromônio e coluna:
            aux = 0;
            for j=1:size(pontos,2)

                % Variável auxiliar que testa se valor de coluna encontrado na
                % matriz feromônio é válido. Caso já seja um nó visitado, é
                % inválido.
                valido = 1;

                % Varrendo linha de matriz de feromônio:
                if matriz_conexao(no_atual,j)==1 && m_feromonio(no_atual,j)>maior

                    % Verificando se valor encontrado de nó já foi visitado. Caso
                    % sim, não é valor válido.
                    for k=1:size(melhor_caminho,2)
                        if j==melhor_caminho(k)
                            valido = 0;
                        end
                    end

                    % Se valor é válido, então atualiza valor do maior e salva
                    % coluna em aux:
                    if valido==1
                        maior = m_feromonio(no_atual,j);
                        aux = j;
                    end
                end
            end

            % Atualizando nó atual, que será o valor da coluna com nível de
            % feromônio maior:
            if aux~=0
                no_atual = aux;
            end

            % Verificando se nó atual é o nó final. Se sim, acionar saída de loop:
            if no_atual==no_final
                sair = 1;
            end

        end % Termina laço que encontra melhor caminho.
    end
end

% Imprimindo o melhor caminho na figura, caso tenha sido encontrado:
if ha_caminho==1 && no_atual==no_final
    % Verificando tamanho do caminho, isto é, coluna do último valor não nulo
    % do vetor de melhor caminho:
    aux = 0;
    for i=1:size(melhor_caminho,2)
        if melhor_caminho(i)~=0
            aux = i;
        end
    end

    if aux>1
        % Imprimindo o caminho encontrado na figura da rede:
        for j=2:aux
            line([pontos(1,melhor_caminho(j-1)) pontos(1,melhor_caminho(j))],...
            [pontos(2,melhor_caminho(j-1)) pontos(2,melhor_caminho(j))],...
            'Color','k','LineStyle','-','LineWidth',2.0);
            hold
        end
    end
end
hold on

%% PARTE 3 - MOVIMENTO DO MÓVEL PELO CAMINHO ENCONTRADO

if ha_caminho==1
    % Simulando o movimento do automóvel:
```

```
% Variáveis a serem utilizadas:
pos_atual = PART;
pos_aux = zeros(2,1);

% Imprimindo o ponto de partida do automóvel:
plot(pos_atual(1,1),pos_atual(2,1),'Marker','.','Color','k','MarkerSize',30);
pause(0.5)

% Percorrendo o melhor caminho e imprimindo na figura:
i = 1;
sair = 0;
direcao = 0;
contador = 0;
while sair==0

    % Atualizando as posições de interesse:
    pos_atual(1,1) = pontos(1,melhor_caminho(i));
    pos_atual(2,1) = pontos(2,melhor_caminho(i));
    pos_aux(1,1) = pontos(1,melhor_caminho(i+1));
    pos_aux(2,1) = pontos(2,melhor_caminho(i+1));

    % Verificando os pontos de acesso ao redor dos extremos do segmento de
    % reta atual:
    v_wfatual = zeros(1,1);
    v_wfaux = zeros(1,1);
    k = 1;
    cont_atual = 1;
    cont_aux = 1;
    while k<=size(pontos,2)
        if distancia(pos_atual(1,1), pos_atual(2,1), pontos(1,k), pontos(2,k))<200
            v_wfatual(cont_atual) = k;
            cont_atual = cont_atual+1;
        end
        if distancia(pos_aux(1,1), pos_aux(2,1), pontos(1,k), pontos(2,k))<200
            v_wfaux(cont_aux) = k;
            cont_aux = cont_aux+1;
        end
        k = k+1;
    end

    % A interseção dos valores desses vetores resultarão nos pontos de
    % acesso mais interessantes, devendo ser escolhidos três deles:
    wifi_disponiveis = zeros(1,1);
    cont = 1;
    for m=1:size(v_wfatual,2)
        for n=1:size(v_wfaux,2)
            if v_wfatual(1,m)==v_wfaux(1,n)
                wifi_disponiveis(cont) = v_wfatual(1,m);
                cont=cont+1;
            end
        end
    end

    % Selecionando três pontos de acesso dentre os disponíveis:
    wifi_ativos = zeros(1,1);

    numeros = randperm(size(wifi_disponiveis,2));
    for k=1:3
        wifi_ativos(1,k) = wifi_disponiveis(1,numeros(k));
    end

    % Imprimido em negrito os pontos de acesso ativos:
    strwf = 'WiFi';
    for m=1:size(wifi_ativos,2)
        text(pontos(1,wifi_ativos(m))+10,pontos(2,wifi_ativos(m))+20,strwf,'Color','k ');
    end

    % Verificando a direção do segmento, isto é, se o automóvel irá
    % percorrer a direção horizontal 1 ou vertical 2:
    if pos_atual(1,1)==pos_aux(1,1)
        direcao = 2;
    else
        direcao = 1;
    end

    % Imprimindo o automóvel percorrendo cada aresta:
    cont = 1;
    while cont<=5
        if direcao==2
            if pos_atual(2,1)<pos_aux(2,1)
                pos_atual(2,1) = pos_atual(2,1)+20;
            else
                pos_atual(2,1) = pos_atual(2,1)-20;
            end
        else
            if pos_atual(1,1)<pos_aux(1,1)
```

```
                    pos_atual(1,1) = pos_atual(1,1)+20;
                else
                    pos_atual(1,1) = pos_atual(1,1)-20;
                end
            end

            plot(pos_atual(1,1),pos_atual(2,1),'Marker','.','Color','k','MarkerSize', 30);
            pause(0.4)
            cont = cont+1;
        end

        % Verificando se o automóvel chegou à posição destino:
        if pos_atual(1,1)==DEST(1,1) && pos_atual(2,1)==DEST(2,1)
            sair = 1;
        end

        % Incrementando i, ou seja, indo para o próximo segmento do caminho a
        % ser percorrido:
        i = i+1;

        % Desligando os pontos de acesso anteriores:
        strwf = 'WiFi';
        for m=1:size(wifi_ativos,2)
            text(pontos(1,wifi_ativos(m))+10,pontos(2,wifi_ativos(m))+20,strwf,'Color','w');
        end

    end
end
```

B – Posicionamento móvel via rede WiFi utilizando o ACO modificado

```
% SISTEMA DE POSICIONAMENTO MÓVEL ACO:

%% Apagando todos os dados/variáveis e fechando janelas:

clear all
close all
clc

%% Declaração das variáveis globais:

global DIMM;    % Dimensão da rede na forma DIMMxDIMM.
global DSUB;    % Dimensão da subrede na forma DIMMxDIMM.
global PART;    % Nó de partida do carro.
global DEST;    % Nó de destino do carro.
global ITER;    % Número de iterações/formigas que passam na rede do ACO.

%% Definição das variáveis globais:

DIMM = 11;
DSUB = 5;
PART = [1000; 1000];
DEST = [0300; 0000];
ITER = 500;

%% Definição de variáveis, vetores e matrizes a serem utilizados durante o programa:

pontos = zeros(2,DIMM^2);       % Cada nó da rede será bem definido na forma [x;y].
matriz_conexao = zeros(DIMM^2); % É uma matriz quadrada que indica conexão entre nós.
matriz_distancia = zeros(1,1);  % Distância entre cada um dos nós da rede.
no_inicial = 0;                 % Coluna da matriz 'pontos' que indica o nó inicial.
no_final = 0;                   % Coluna da matriz 'pontos' que indica o nó final.
coef_angular = 0;               % Coeficiente angular da reta entre nó inicial e final.
no_atual = 0;                   % Indica o movimento intermediário do automóvel.
pontos_sub = zeros(2,DSUB^2);   % Número máximo de pontos da subrede.
m_conexao_sub = zeros(DSUB^2);  % Matriz que indica conexão entre os nós da subrede.
quadrante = 0;                  % Quadrante relativo entre pontos atual e final.
posicao = 0;                    % Mostra o a posição relativa entre pontos atual e final.
xf_ideal = 0;                   % Abcissa final ideal de cada subrede.
yf_ideal = 0;                   % Ordenada final ideal de cada subrede.
xf_sub = 0;                     % Abcissa final de cada subrede.
yf_sub = 0;                     % Ordenada final de cada subrede.
no_final_sub = 0;               % Indica o nó final a cada subrede.
v_visitados_sub = zeros(1,1);   % Vetor que mostra nós já visitados por cada formiga.
v_caminho_sub = zeros(1,1);     % Caminho percorrido por cada formiga.
no_aux = 0;                     % Indica o nó atual de cada formiga ao percorrer uma subrede.
vetor_saidas = zeros(1,1);      % Mostra as opções de arestas a serem seguidas a cada nó.
saidas = 0;                     % Número de possíveis saídas a cada nó.
distancia_total_sub = 0;        % Distância percorrida por cada formiga.
melhor_caminho_sub = zeros(1,1);% Menor caminho encontrado pelas formigas na subrede.
melhor_caminho = zeros(1,1);    % Melhor caminho global encontrado.

%% Definição de variáveis auxiliares e flags:

contador = 0;
i = 0;
j = 0;
k = 0;
sair = 0;
indice = 0;
flag_saida = 0;
ha_caminho = 0;
dado = 0;
m = 0;
n = 0;

%% Definição de strings:

str_partida = 'PART';
str_destino = 'DEST';
str_wifi = 'WIFI';

%% Definição dos pontos da rede:

% Cada ponto é fisicamente um nó da rede e vão de 1 a DIMM^2.
contador = 1;
for j=1:DIMM
    for i=1:DIMM
        pontos(1,contador) = (i-1)*100;
        pontos(2,contador) = (j-1)*100;
        contador=contador+1;
```

```
    end
end

%% Definição da matriz conexão:

% Essa matriz define se um nó da linha i tem conexão com o nó da coluna j.
% É definido 1 se há e 0 se não há conexão.
for i=1:size(pontos, 2)
    if i==1
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i+DIMM)=1;
    elseif i>1 && i<DIMM
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i+DIMM)=1;
    elseif i==DIMM
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i+DIMM)=1;
    elseif i>1 && mod((i-1),DIMM)==0 && i<(DIMM^2-DIMM+1)
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-DIMM)=1;
        matriz_conexao(i,i+DIMM)=1;
    elseif i>DIMM && mod(i,DIMM)==0 && i<DIMM^2
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i-DIMM)=1;
        matriz_conexao(i,i+DIMM)=1;
    elseif i==(DIMM^2-DIMM+1)
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-DIMM)=1;
    elseif i>(DIMM^2-DIMM+1) && i<DIMM^2
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-DIMM)=1;
    elseif i==DIMM^2
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i-DIMM)=1;
    else
        matriz_conexao(i,i-1)=1;
        matriz_conexao(i,i+1)=1;
        matriz_conexao(i,i-DIMM)=1;
        matriz_conexao(i,i+DIMM)=1;
    end
end

%% Matriz distância da rede:

% Essa matriz indica a distância entre cada um dos nós e é simétrica.
matriz_distancia = zeros(size(pontos,2));
for i=1:size(pontos,2)
    for j=i:size(pontos,2)
        matriz_distancia(i,j) = distancia(pontos(1,i),pontos(2,i),pontos(1,j),pontos(2,j));
        matriz_distancia(j,i) = matriz_distancia(i,j);
    end
end

%% Encontrando os pontos de partida e de destino:

% Encontrando o ponto inicial escolhido para o carro, isto é, a coluna da
% matriz 'pontos' que indica o ponto de partida na rede:
for i=1:size(pontos,2)
    if PART(1)==pontos(1,i) && PART(2)==pontos(2,i)
        no_inicial = i;
        break;
    end
end

% Encontrando o ponto final escolhido para o carro, isto é, a coluna da
% matriz 'pontos' que indica o ponto de destino na rede:
for i=1:size(pontos,2)
    if DEST(1)==pontos(1,i) && DEST(2)==pontos(2,i)
        no_final = i;
        break;
    end
end

%% Imprimindo a rede:

figure('units','normalized','outerposition',[0 0 .6 1])
for i=1:size(matriz_conexao,1)
    for j=i:size(matriz_conexao,2)
        if matriz_conexao(i,j)==1
            line([pontos(1,i) pontos(1,j)],[pontos(2,i) pontos(2,j)],...
            'Color','k','LineStyle','--','LineWidth',0.2);
            hold;
        end
    end
```

```
end
% Eixos do gráfico:
% x: -50 a DIMM*100-50.
% y: -50 a DIMM*100-50.
axis([-50 DIMM*100-50 -50 DIMM*100-50]);

% Imprimindo o ponto de partida na figura:
text(pontos(1,no_inicial)+10,pontos(2,no_inicial)-15,str_partida);
hold on;
plot(pontos(1,no_inicial),pontos(2,no_inicial),...
'Marker','.','Color',[.75 .75 .75],'MarkerSize',30);

% Imprimindo o ponto de destino na figura:
text(pontos(1,no_final)+10,pontos(2,no_final)-15,str_destino);
hold on;
plot(pontos(1,no_final),pontos(2,no_final),...
'Marker','.','Color',[.75 .75 .75],'MarkerSize',30);

% Encontrando os coeficientes da equação de reta entre pontos inicial e final:
coef_angular = (pontos(2,no_final)-pontos(2,no_inicial))/(pontos(1,no_final)-pontos(1,no_inicial));

%% Encontrando o melhor caminho:

% Cronometrando tempo de busca:
tic;

% Será encontrado de subrede a subrede até o ponto de destino.
no_atual = no_inicial;
sair = 0;
indice = 1;
melhor_caminho(indice) = no_inicial;

% Começo do laço que encontra o melhor caminho:
while sair==0
    %% Zerando variáveis:

    quadrante = 0;
    contador = 0;
    pontos_sub = zeros(2,DSUB^2);
    m_conexao_sub = zeros(DSUB^2);
    posicao = 0;
    no_final_sub = 0;

    %% Checando o quadrante relativo entre ponto atual e ponto final:

    if pontos(2,no_final)>=pontos(2,no_atual) && pontos(1,no_final)>=pontos(1,no_atual)
        quadrante = 1;
        sinal_x = +1;
        sinal_y = +1;
    elseif pontos(2,no_final)>pontos(2,no_atual) && pontos(1,no_final)<pontos(1,no_atual)
        quadrante = 2;
        sinal_x = -1;
        sinal_y = +1;
    elseif pontos(2,no_final)<=pontos(2,no_atual) && pontos(1,no_final)<=pontos(1,no_atual)
        quadrante = 3;
        sinal_x = -1;
        sinal_y = -1;
    elseif pontos(2,no_final)<pontos(2,no_atual) && pontos(1,no_final)>pontos(1,no_atual)
        quadrante = 4;
        sinal_x = +1;
        sinal_y = -1;
    end

    %% Definindo os pontos da subrede:

    switch quadrante
        % Quadrante = 1?
        case 1
            contador = 1;
            for j=1:DSUB
                for i=1:DSUB
                    pontos_sub(1,contador) = pontos(1,no_atual)+(i-1)*100;
                    pontos_sub(2,contador) = pontos(2,no_atual)+(j-1)*100;
                    contador=contador+1;
                end
            end
        % Quadrante = 2?
        case 2
            contador = 1;
            for j=1:DSUB
                for i=DSUB:-1:1
                    pontos_sub(1,contador) = pontos(1,no_atual)+(1-i)*100;
                    pontos_sub(2,contador) = pontos(2,no_atual)+(j-1)*100;
                    contador=contador+1;
                end
            end
```

```
    % Quadrante = 3?
    case 3
        contador = 1;
        for j=DSUB:-1:1
            for i=DSUB:-1:1
                pontos_sub(1,contador) = pontos(1,no_atual)+(1-i)*100;
                pontos_sub(2,contador) = pontos(2,no_atual)+(1-j)*100;
                contador=contador+1;
            end
        end
    % Quadrante = 4?
    case 4
        contador = 1;
        for j=DSUB:-1:1
            for i=1:DSUB
                pontos_sub(1,contador) = pontos(1,no_atual)+(i-1)*100;
                pontos_sub(2,contador) = pontos(2,no_atual)+(1-j)*100;
                contador=contador+1;
            end
        end
end

%% Definindo uma matriz conexão da subrede:

for i=1:size(pontos_sub,2)
    if i==1
        m_conexao_sub(i,i+1)=1;
        m_conexao_sub(i,i+DSUB)=1;
    elseif i>1 && i<DSUB
        m_conexao_sub(i,i+1)=1;
        m_conexao_sub(i,i-1)=1;
        m_conexao_sub(i,i+DSUB)=1;
    elseif i==DSUB
        m_conexao_sub(i,i-1)=1;
        m_conexao_sub(i,i+DSUB)=1;
    elseif i>1 && mod((i-1),DSUB)==0 && i<(DSUB^2-DSUB+1)
        m_conexao_sub(i,i+1)=1;
        m_conexao_sub(i,i-DSUB)=1;
        m_conexao_sub(i,i+DSUB)=1;
    elseif i>DSUB && mod(i,DSUB)==0 && i<DSUB^2
        m_conexao_sub(i,i-1)=1;
        m_conexao_sub(i,i-DSUB)=1;
        m_conexao_sub(i,i+DSUB)=1;
    elseif i==(DSUB^2-DSUB+1)
        m_conexao_sub(i,i+1)=1;
        m_conexao_sub(i,i-DSUB)=1;
    elseif i>(DSUB^2-DSUB+1) && i<DSUB^2
        m_conexao_sub(i,i-1)=1;
        m_conexao_sub(i,i+1)=1;
        m_conexao_sub(i,i-DSUB)=1;
    elseif i==DSUB^2
        m_conexao_sub(i,i-1)=1;
        m_conexao_sub(i,i-DSUB)=1;
    else
        m_conexao_sub(i,i-1)=1;
        m_conexao_sub(i,i+1)=1;
        m_conexao_sub(i,i-DSUB)=1;
        m_conexao_sub(i,i+DSUB)=1;
    end
end

%% Imprimindo a subrede:

for i=1:DSUB^2
    for j=i:DSUB^2
        if m_conexao_sub(i,j)==1
            line([pontos_sub(1,i) pontos_sub(1,j)],[pontos_sub(2,i) pontos_sub(2,j)],...
            'Color','r','LineStyle','-','LineWidth',1.0);
            hold;
        end
    end
end

%% Definindo nó posição relativa entre nó atual e nó final:

% Se ponto final se encontra dentro da subrede atual:
if abs(pontos(2,no_final)-pontos(2,no_atual))<=100*(DSUB-1) && ...
   abs(pontos(1,no_final)-pontos(1,no_atual))<=100*(DSUB-1)
    posicao = 1;
% Se ponto final se encontra fora da subrede atual:
else
    posicao = 2;
end

%% Definindo destino temporário de acordo com a posição:
```

```
    switch posicao
        % No caso 1, o ponto final se encontra dentro da subrede:
        case 1
            for i=1:size(pontos_sub,2)
                if pontos_sub(1,i)==pontos(1,no_final) && pontos_sub(2,i)==pontos(2,no_final)
                    no_final_sub = i;
                end
            end
        % No caso 2, o ponto final se encontra fora da subrede:
        case 2
            % Pontos em que a reta cartesiana corta as bordas da subrede:
            xf_ideal = pontos(1,no_atual)+sinal_x*100*(DSUB-1);
            yf_ideal = pontos(2,no_atual)+sinal_y*100*(DSUB-1);

            if pontos(1,no_atual)==pontos(1,no_final)
                x_estrela = pontos(1,no_atual);
            else
                x_estrela = (yf_ideal-
pontos(2,no_inicial)+coef_angular*pontos(1,no_inicial))/coef_angular;
            end

            if pontos(2,no_atual)==pontos(2,no_final)
                y_estrela = pontos(2,no_atual);
            else
                y_estrela = coef_angular*(xf_ideal-pontos(1,no_inicial))+pontos(2,no_inicial);
            end

            % Definindo pontos de extremidade da subrede:
            vx = [pontos(1,no_atual)+sinal_x*100*(DSUB-1); pontos(2,no_atual); 0];
            vy = [pontos(1,no_atual); pontos(2,no_atual)+sinal_y*100*(DSUB-1); 0];
            vz = [pontos(1,no_atual)+sinal_x*100*(DSUB-1); pontos(2,no_atual)+sinal_y*100*(DSUB-1);
0];

            if abs(y_estrela-pontos(2,no_atual))<=100*(DSUB-1)
                xf_sub = xf_ideal;
                for k=1:DSUB
                    if distancia(xf_sub,pontos(2,no_atual),xf_sub,pontos(2,no_atual)+sinal_y*100*(k-
1))-...
                       distancia(xf_sub,pontos(2,no_atual),xf_sub,y_estrela)<=0
                        yf_sub = pontos(2,no_atual)+sinal_y*100*(k-1);
                    end
                end
            elseif abs(x_estrela-pontos(1,no_atual))<=100*(DSUB-1)
                yf_sub = yf_ideal;
                for k=1:DSUB
                    if distancia(pontos(1,no_atual),yf_sub,pontos(1,no_atual)+sinal_x*100*(k-
1),yf_sub)-...
                       distancia(pontos(1,no_atual),yf_sub,x_estrela,yf_sub)<=0
                        xf_sub = pontos(1,no_atual)+sinal_x*100*(k-1);
                    end
                end
            end

            % Definindo nó final da subrede:
            for i=1:size(pontos_sub,2)
                if pontos_sub(1,i)==xf_sub && pontos_sub(2,i)==yf_sub
                    no_final_sub = i;
                end
            end
    end

    %% Imprimindo destino temporário:
    plot(pontos_sub(1,no_final_sub),pontos_sub(2,no_final_sub),...
    'Marker','.','Color','k','MarkerSize',30);

    %% Aplicação do ACO na subrede:

    % Atribuindo um valor teto para menor distância encontrada na subrede:
    menor_distancia_sub = 100*DSUB^2;

    % Busca do melhor caminho:
    ha_caminho = 0;
    while ha_caminho==0

        for i=1:1:ITER

            % Atribuindo valor a contador auxiliar:
            contador = 1;

            % Inicializando um vetor que indica quais nós ainda foram ou não
            % visitados. Se nó da coluna j já foi visitado, então recebe valor 0.
            % Caso contrário, recebe 1:
            v_visitados_sub = ones(1,size(pontos_sub,2));

            % Inicializando um vetor auxiliar para indicar o caminho percorrido
            % por cada formiga:
```

```
v_caminho_sub = zeros(1,1);

% O nó atual indica onde cada formiga se encontra a cada movimento na
% rede:
for j=1:size(pontos_sub,2)
    if pontos_sub(1,j)==pontos(1,no_atual) && ...
       pontos_sub(2,j)==pontos(2,no_atual)
        no_atual_sub = j;
    end
end

% O primeiro valor do caminho a ser percorrido é o próprio ponto de
% partida:
v_caminho_sub(1,1) = no_atual_sub;

% Inicializando flag para saída de loop e flag de verificação se
% uma formiga encontrou ou não um caminho:
flag_saida = 0;
ha_caminho = 0;
while flag_saida==0

    % Se o nó atual for diferente do nó final, então continua
    % procurando um caminho:
    if no_atual_sub~=no_final_sub || sair~=1

        % Inicializando o vetor linha, cujo conteúdo indica os
        % caminhos, i.e, pontos possíveis de se seguir a cada nó:
        vetor_saidas = zeros(1,4);

        % O primeiro nó do caminho a ser percorrido, i.e, o nó inicial
        % é um nó visitado:
        v_visitados_sub(v_caminho_sub(contador))=0;
        contador = contador+1;

        % Inicializando uma variável que mostra o número total de
        % possíveis saídas do nó atual:
        saidas = 0;

        % Verificando as possíveis saídas para o nó atual:
        for j=1:size(pontos_sub,2)

            % Verificando quantas e quais saídas são as possíveis para
            % o nó atual. Se uma das possíveis saídas é um nó já
            % visitado, essa possibilidade não é contabilizada através
            % da multiplicação pelo vetor de nós visitados:
            if m_conexao_sub(no_atual_sub,j)*v_visitados_sub(j)==1
                saidas = saidas+1;
                vetor_saidas(saidas) = j;
            end
        end

        % Sorteando uma saída aleatoriamente para cada caso:
        switch saidas
            % Se não houver mais saídas, então não há mais caminho a
            % pecorrer
            case 0
                flag_saida = 1;
            % Caso o número de saídas seja um, então há apenas um
            % caminho a seguir.
            case 1
                v_caminho_sub(contador) = vetor_saidas(1);
            % Caso houver duas possíveis saídas, a variável dado é
            % acionada e, se for menor que 5, a primeira saída do vetor
            % linha de saídas é selecionada. Caso contrário, a segunda
            % saída possível é selecionada.
            case 2
                dado = randperm(2,1);
                if dado==1
                    v_caminho_sub(contador) = vetor_saidas(1);
                else
                    v_caminho_sub(contador) = vetor_saidas(2);
                end
            % Caso o número de saídas possíveis sejam três, um sorteio
            % é feito semelhante ao caso anterior, porém com as
            % possibilidades divididas em três.
            case 3
                dado = randperm(3,1);
                if dado==1
                    v_caminho_sub(contador) = vetor_saidas(1);
                elseif dado==2
                    v_caminho_sub(contador) = vetor_saidas(2);
                else
                    v_caminho_sub(contador) = vetor_saidas(3);
                end
            % Caso o número de saídas possíveis sejam quatro, um sorteio
            % é feito com as possibilidades divididas em quatro.
```

```
                        case 4
                            dado = randperm(4,1);
                            if dado==1
                                v_caminho_sub(contador) = vetor_saidas(1);
                            elseif dado==2
                                v_caminho_sub(contador) = vetor_saidas(2);
                            elseif dado==3
                                v_caminho_sub(contador) = vetor_saidas(3);
                            else
                                v_caminho_sub(contador) = vetor_saidas(4);
                            end
                        otherwise
                    end % Fim do switch com variável saída.

                    % Atualizando nó atual:
                    if flag_saida==0
                        no_atual_sub = v_caminho_sub(contador);
                    end

                    % Checando se o nó atual é o final:
                    if no_atual_sub==no_final_sub
                        flag_saida = 1;
                        ha_caminho = 1;
                    end

                end % Fim da condição if com variável flag.

            end % Fim do laço para encontrar ou não um caminho.

            % Se foi encontrado um caminho cujos nós inicial e final sejam os
            % desejados, então há cálculo da distância desse caminho encontrado a
            % cada formiga que passa na rede. Ao final de todas iterações/formigas
            % que passam pela rede, a menor distância será guardade, indicando o
            % menor caminho:

            % Calculando a distância a cada iteração:
            if ha_caminho==1
                distancia_total_sub = 0;
                % Encontrando distância total do caminho percorrido, caso seja de
                % interesse:
                for j=2:size(v_caminho_sub,2)
                    if v_caminho_sub(j)~=0
                        distancia_total_sub = distancia_total_sub + ...
                        distancia(pontos_sub(1,v_caminho_sub(j-1)),pontos_sub(2,v_caminho_sub(j-
1)),...
                        pontos_sub(1,v_caminho_sub(j)),pontos_sub(2,v_caminho_sub(j)));
                    end
                end

                % Se o caminho encontrado tiver uma distância menor entre o ponto
                % de partida e destino, então será o melhor caminho:
                if distancia_total_sub<menor_distancia_sub
                    menor_distancia_sub = distancia_total_sub;
                    melhor_caminho_sub = zeros(1,1);
                    for j=1:size(v_caminho_sub,2)
                        melhor_caminho_sub(j) = v_caminho_sub(j);
                    end
                end
            end

        end % Final da iteração do número de formigas.

    end % Final do laço do ACO.

    %% Salvando o melhor caminho no vetor global:

    for m=2:size(melhor_caminho_sub,2)
        for n=1:size(pontos,2)
            if pontos_sub(1,melhor_caminho_sub(1,m))==pontos(1,n) && ...
                    pontos_sub(2,melhor_caminho_sub(1,m))==pontos(2,n)
                indice = indice+1;
                melhor_caminho(indice) = n;
            end
        end
    end

    %% Atualizando o valor do nó atual:
    for i=1:size(pontos,2)
        if pontos_sub(1,no_final_sub)==pontos(1,i) && ...
           pontos_sub(2,no_final_sub)==pontos(2,i)
            no_atual = i;
        end
    end

    % Verificando tamanho do caminho, isto é, coluna do último valor não nulo
    % do vetor de melhor caminho:
```

```
    aux = 0;
    for i=1:size(melhor_caminho_sub,2)
        if melhor_caminho_sub(i)~=0
            aux = i;
        end
    end

    if aux>1
        % Imprimindo o caminho encontrado na figura da rede:
        for j=2:aux
            line([pontos_sub(1,melhor_caminho_sub(j-1)) pontos_sub(1,melhor_caminho_sub(j))],...
                [pontos_sub(2,melhor_caminho_sub(j-1)) pontos_sub(2,melhor_caminho_sub(j))],...
                'Color','k','LineStyle','-','LineWidth',2.0);
            hold;
        end
    end
    hold on;

    %% Controle de saída do laço que encontra o melhor caminho:
    if no_atual==no_final
        sair = 1;
    end

end % Fim do laço que encontra o melhor caminho.
tempo_de_busca = toc;

distancia_total = 0;
% Encontrando distância total do caminho percorrido, caso seja de
% interesse:
for j=2:size(melhor_caminho,2)
    distancia_total = distancia_total + ...
    distancia(pontos(1,melhor_caminho(j-1)),pontos(2,melhor_caminho(j-1)),...
    pontos(1,melhor_caminho(j)),pontos(2,melhor_caminho(j)));
end

%% Movimento do móvel pelo caminho encontrado:

if ha_caminho==1
    % Simulando o movimento do automóvel:
    % Variáveis a serem utilizadas:
    pos_atual = PART;
    pos_aux = zeros(2,1);

    % Imprimindo o ponto de partida do automóvel:
    plot(pos_atual(1,1),pos_atual(2,1),'Marker','.','Color','k','MarkerSize',30);
    pause(0.5)

    % Percorrendo o melhor caminho e imprimindo na figura:
    i = 1;
    sair = 0;
    direcao = 0;
    contador = 0;
    while sair==0

        % Atualizando as posições de interesse:
        pos_atual(1,1) = pontos(1,melhor_caminho(i));
        pos_atual(2,1) = pontos(2,melhor_caminho(i));
        pos_aux(1,1) = pontos(1,melhor_caminho(i+1));
        pos_aux(2,1) = pontos(2,melhor_caminho(i+1));

        % Verificando os pontos de acesso ao redor dos extremos do segmento de
        % reta atual:
        v_wfatual = zeros(1,1);
        v_wfaux = zeros(1,1);
        k = 1;
        cont_atual = 1;
        cont_aux = 1;
        while k<=size(pontos,2)
            if distancia(pos_atual(1,1), pos_atual(2,1), pontos(1,k), pontos(2,k))<200
                v_wfatual(cont_atual) = k;
                cont_atual = cont_atual+1;
            end
            if distancia(pos_aux(1,1), pos_aux(2,1), pontos(1,k), pontos(2,k))<200
                v_wfaux(cont_aux) = k;
                cont_aux = cont_aux+1;
            end
            k = k+1;
        end

        % A interseção dos valores desses vetores resultarão nos pontos de
        % acesso mais interessantes, devendo ser escolhidos três deles:
        wifi_disponiveis = zeros(1,1);
        cont = 1;
        for m=1:size(v_wfatual,2)
            for n=1:size(v_wfaux,2)
                if v_wfatual(1,m)==v_wfaux(1,n)
```

```
                    wifi_disponiveis(cont) = v_wfatual(1,m);
                    cont=cont+1;
                end
            end
        end

        % Selecionando três pontos de acesso dentre os disponíveis:
        wifi_ativos = zeros(1,1);

        numeros = randperm(size(wifi_disponiveis,2));
        for k=1:3
            wifi_ativos(1,k) = wifi_disponiveis(1,numeros(k));
        end

        % Imprimido em negrito os pontos de acesso ativos:
        strwf = 'WiFi';
        for m=1:size(wifi_ativos,2)
            text(pontos(1,wifi_ativos(m))+10,pontos(2,wifi_ativos(m))+20,strwf,'Color','k ');
        end

        % Verificando a direção do segmento, isto é, se o automóvel irá
        % percorrer a direção horizontal 1 ou vertical 2:
        if pos_atual(1,1)==pos_aux(1,1)
            direcao = 2;
        else
            direcao = 1;
        end

        % Imprimindo o automóvel percorrendo cada aresta:
        cont = 1;
        while cont<=5
            if direcao==2
                if pos_atual(2,1)<pos_aux(2,1)
                    pos_atual(2,1) = pos_atual(2,1)+20;
                else
                    pos_atual(2,1) = pos_atual(2,1)-20;
                end
            else
                if pos_atual(1,1)<pos_aux(1,1)
                    pos_atual(1,1) = pos_atual(1,1)+20;
                else
                    pos_atual(1,1) = pos_atual(1,1)-20;
                end
            end

            plot(pos_atual(1,1),pos_atual(2,1),'Marker','.','Color','k','MarkerSize', 30);
            pause(0.4)
            cont = cont+1;
        end

        % Verificando se o automóvel chegou à posição destino:
        if pos_atual(1,1)==DEST(1,1) && pos_atual(2,1)==DEST(2,1)
            sair = 1;
        end

        % Incrementando i, ou seja, indo para o próximo segmento do caminho a
        % ser percorrido:
        i = i+1;

        % Desligando os pontos de acesso anteriores:
        strwf = 'WiFi';
        for m=1:size(wifi_ativos,2)
            text(pontos(1,wifi_ativos(m))+10,pontos(2,wifi_ativos(m))+20,strwf,'Color','w');
        end

    end
end
```