Definição do Trajeto de um Robô Inspetor de Vasos de Pressão Esféricos Utilizando Otimização por Colônia de Formigas

Nicolas Dalmedico, André Schneider de Oliveira, Heitor Silvério Lopes
Universidade Tecnológica Federal do Paraná
Av. 7 de Setembro, 3165 - 80230-901- Curitiba (PR)
E-mails: ndalmedico@alunos.utfpr.edu.br, {andreoliveira,hslopes}@utfpr.edu.br

***Resumo*—Tanques esféricos de armazenamento de derivados de petróleo requerem inspeção para garantir sua integridade, uma vez que são constituídos de placas soldadas cujas trilhas de solda podem desgastar e causar acidentes com consequências devastadoras. O desenvolvimento de um robô autônomo inspetor de cordões de solda tem o objetivo de tornar o processo mais rápido e diminuir o risco de erro humano. Entretanto, para que o robô realize uma inspeção completa, é necessário definir uma trajetória que será percorrida dentro do tanque de armazenamento. O problema de cobrir todas trilhas de solda se assemelha ao Problema do Caixeiro Viajante. Este artigo apresenta um algoritmo que calcula uma trajetória íntegra passando por todos as trilhas de solda utilizando otimização por colônia de formigas para definir a menor trajetória que cumpra o objetivo. Comparações são apresentadas em relação a outros solucionadores e outras abordagens para validação.**

***Palavras-chave*—Otimização; Colônia de Formiga; Robô móvel; Planejamento de Trajetória; Inspeção**

I. Introdução

Tanques de esféricos armazenamento, como o mostrado na Figura 1, são equipamentos utilizados para o armazenamento de grandes quantidades de produtos como o petróleo e seus derivados, produtos químicos, resíduos, algumas misturas e água. A sua integridade é de extrema importância para a indústria, pois caso seja prejudicada, pode causar desastres ambientais graves e custosos. Por esta razão há a necessidade de monitoramento contínua dos tanques para a avaliação dos processos de corrosão.

Existem diversas técnicas para a inspeção dos tanques de armazenamento, sendo que a mais comum dentre elas é o uso de ultrassom manual. Neste caso, um operador realiza a verificação do tanque de armazenamento passando o ultrassom por todo tanque. No entanto, além dos riscos ao operador, este procedimento requer um sistema eficiente de escalada para que o operador possa percorrer toda a área do tanque. Por estes motivos um robô de inspeção de trilhas de solda tem grande importância para indústria do petróleo. O algoritmo apresentado neste artigo é orientado ao robô descrito em [2]–[4]. Este robô inspetor de tanques esféricos utiliza rodas magnéticas para aderir à parede do tanque e um transdutor ultrassom em sua parte inferior para realizar a inspeção da solda.

Para facilitar o processo de inspeção, o robô seguidor de solda requer autonomia dentro do espaço de inspeção, isto é, o interior do tanque. Para tanto, o robô conta com vários sensores espaciais que lhe permitem conhecer sua própria posição, identificar obstáculos e detectar cordões de solda. Entretanto, para uma inspeção rigorosa, é importante que o robô tenha uma trajetória precisamente definida, que o permita cobrir toda a extensão de cordões de solda do tanque, percorrendo a menor distância possível. Os algoritmos apresentados neste artigo têm o objetivo de definir tal trajetória a partir da localização dos cordões de solda e seus pontos de intersecção, utilizando o método da Otimização por Colônia de Formigas (*Ant Colony Optimization* – ACO).

Figura 1: Tanque esférico de armazenamento de derivados de petróleo [1].

II. Revisão da Literatura

Na área de Computação Evolucionária, há diversos algoritmos consagrados, sendo alguns mais específicos e outros

mais genéricos. Porém, todos são voltados a problemas de otimização, especialmente nas áreas de Engenharia e de Pesquisa Operacional [5]. Os algoritmos evolucionários utilizam conceitos provenientes do princípio de seleção natural para abordar problemas de otimização. Eles são robustos, genéricos e facilmente adaptáveis a problemas de diversas naturezas, notadamente em Engenharia. Dentre os principais algoritmos evolucionários estão os Algoritmos Genéticos (AG), a Programação Genética (PG), a Otimização por Colônias de Formigas (ACO) e a Otimização por Enxame de Partículas (PSO).

A utilização de algoritmos evolucionários no planejamento de rotas já é um assunto bem estudado. Em [6] é utilizado um AG o planejamento de menor rota em ambiente com obstáculos. Em [7] é realizada determinação de rota com múltiplos objetivos para veículos aéreos não-tripulados. Em [8] é realizado planejamento de rota de múltiplos objetivos por AG levando em consideração a dificuldade para se percorrer cada trecho.

O método utilizado neste trabalho é o ACO [9], uma metaheurística baseada no comportamento de formigas quando buscam por alimento. ACO também tem sido muito utilizado para problemas específicos de roteamento. Em [10] é reportado um ACO para o roteamento ótimo multiobjetivo em redes veiculares ad-hoc. Em [11] também é utilizado um ACO, mas para o roteamento de veículos entre múltiplos depósitos. Em [12] é abordado o problema de roteamento de veículos heterogêneos (motocicletas e caminhões) em uma rede de distribuição de produtos. De maneira similar, [13] abordaram roteamento e *scheduling* de frotas heterogêneas com ACO. Em [14], [15] são reportados algoritmos híbridos entre *fuzzy*+ACO e AG+ACO para planejamento de rotas.

III. Métodos

No presente trabalho, duas abordagens foram definidas para testes e avaliação no quesito tempo de execução e tamanho da melhor rota, sendo a segunda abordagem a melhor contribuição, desenvolvida em cima das lições aprendidas pela implementação da primeira abordagem. A primeira abordagem utiliza de punição para convergência da rota e para maximizar o número de trilhas de solda percorrida. Nesta abordagem as trilhas de solda se encontram discretizadas em vários pontos. O segundo algoritmo não utiliza punição nem trilhas discretizadas. Ao invés disto garante que cada solução encontrada tenha percorrido todo o tanque, e as trilhas se encontram representadas apenas pelas suas intersecções.

Uma comparação do tempo de convergência para uma solução ótima será realizada. Uma solução ótima percorre todas as trilhas de solda e faz isto percorrendo menos de 700m dentro do tanque, um valor empírico definido baseado nos resultados dos algoritmos testados.

O algoritmo ACO implementado simula formigas artificiais que saem de um ponto inicial e seguem um, dentre vários caminhos, baseando-se na quantidade de feromônio presente em cada caminho. O feromônio é depositado nas trilhas proporcionalmente à distância percorrida no final de cada simulação, e evapora em função da distância para facilitar a convergência.

O algoritmo de otimização por colônia de formigas possui vários parâmetros que definem seu modelo. O algoritmo gera um número determinado 'nAnt' de formigas que buscam rotas e deixam feromônios onde passam. Este feromônio evapora em uma taxa 'rho', para que na próxima iteração as formigas possam decidir seguir os caminhos baseando-se na quantidade de feromônio deixado, que será proporcional a distância do caminho percorrido. O número de iterações 'MaxIt' define quantas vezes este processo se repete. No momento de decisão para qual ponto se mover, dois fatores são levados em consideração, a distância e o feromônio presente. Estes fatores são ponderados exponencialmente por dois parâmetros, 'alfa' para o feromônio, e 'beta' para a distância. Isso gera uma matriz de pesos que define a probabilidade da formiga escolher o próximo ponto. A escolha é feita por torneio, onde um número 'tournament' de pontos é sorteado e o melhor ponto é escolhido.

As primeiras formigas possuem comportamento absolutamente aleatório, uma vez que a quantidade de feromônio inicial é igual em todos os caminhos. Porém, à medida que novas formigas são geradas, os caminhos mais curtos acumulam mais feromônio e, assim, as próximas formigas são atraídas para estas trilhas com mais frequência.

Para o desenvolvimento do algoritmo foi utilizado o software MATLAB, que é uma plataforma muito popular para o desenvolvimento de aplicações científicas. A principal biblioteca para MATLAB utilizada como base para os programas está disponível em [16]. Outra biblioteca utilizada para a manipulação de formas geométricas em MATLAB foi a Geom3D [17].

A. Primeira Abordagem

A primeira versão do algoritmo desenvolvido aborda os cordões de solda discretizados em diversos pontos. Neste caso, o problema a ser resolvido se torna similar ao problema clássico do caixeiro viajante.

Um tanque esférico de armazenamento de derivados de petróleo foi simulado no MATLAB e suas trilhas de solda foram discretizadas a cada 1 grau, conforme mostrado na Figura 2. Os dados para reconstrução e discretização do tanque são provenientes do algoritmo identificador de tanques desenvolvido em [3]. Os pontos gerados pela discretização foram armazenados em três vetores representando suas coordenadas x, y, z. A seguir, os pontos foram convertidos para coordenadas esféricas, de forma a simplificar o problema, uma vez que o raio constante da esfera pôde ser removido dos cálculos. Este artifício permitiu reduzir o problema de três para duas dimensões. Os dois novos vetores gerados, azimute e elevação, permitem a aplicação direta do algoritmo ACO.

O algoritmo cria um número predeterminado de formigas no ponto de início do espaço de busca, gera uma trajetória para cada uma delas, atualiza a matriz de feromônios, e avalia a melhor formiga baseando-se na distância percorrida. Este procedimento é repetido por um número determinado de

iterações. Para a determinação da distância, é utilizada uma matriz de distâncias entre os pontos de interesse.

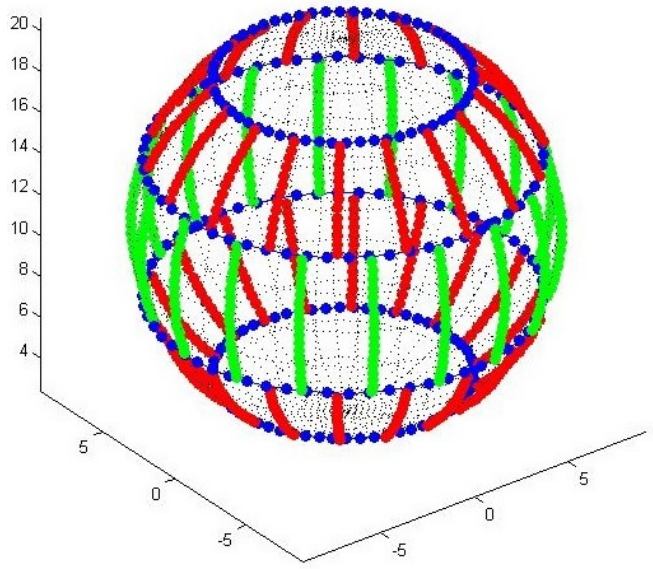

Figura 2: Representação do tanque esférico e seus cordões de solda.

A matriz de distâncias entre os pontos foi gerada utilizando-se a função *distance*, que calcula a distância em arco radiano entre dois pontos de uma esfera e o ângulo do arco entre os pontos em relação ao horizonte da esfera. A função permite driblar dois possíveis problemas do algoritmo: extrapolar o limite da matriz de distância (uma vez que o MATLAB sempre calcula a menor distância entre dois pontos, assumindo o maior valor possível como π) e calcular o ângulo assumido entre o arco entre os pontos em relação ao horizonte da esfera, dado que virá a ser importante na aplicação de penalidades.

Uma matriz de penalidades foi implementada junto com a geração da matriz de distâncias, possuindo tamanho igual à mesma. O objetivo desta matriz foi aplicar penalidades para caminhos que não eram interessantes para a solução do problema. A ideia desta matriz foi inspirada em um procedimento similar, bastante comum em AGs, proposto por Goldberg [18].

A primeira penalização é aplicada a novos caminhos com ângulos que não são retos, uma vez que, observando a disposição das trilhas de solda, é possível perceber que soluções próximas do ideal não requerem que as formigas tracem caminhos na diagonal. A segunda penalização é aplicada a pontos localizados a mais de 6 graus de distância do atual, para incentivar as formigas a cobrirem os pontos mais próximos, acelerando a convergência.

Para acelerar a execução do algoritmo e facilitar a convergência, os pontos foram selecionados de modo a ficarem quase igualmente espaçados. No total, foram amostrados 504 pontos para o tanque esférico simulado. O restante dos parâmetros foram ajustados empiricamente com testes preliminares, sendo a única alteração considerável a taxa de evaporação de feromônio (*rho*), aumentada de 5% para 25%, para penalizar ainda mais rotas que incluam caminhos longos. A seleção do próximo ponto para cada formiga é feita pelo método do torneio estocástico, procedimento igualmente inspirado em AGs. No torneio, um número previamente estabelecido de pontos é aleatoriamente selecionado e o ponto com mais feromônio (consequentemente o mais próximo) é escolhido.

O algoritmo apresentou um tempo consideravelmente alto de execução, levando em torno de 1 segundo por iteração do *loop* principal, totalizando cerca de 3 minutos. Independente dos parâmetros ou penalizações aplicadas, o algoritmo não apresentou resultado satisfatório, pois não conseguiu garantir que a cobertura completa dos cordões de solda. A melhor solução obtida por esta abordagem levou a uma distância total percorrida de 613 metros (esfera com raio de 9,125 metros). A trajetória mais próxima do ideal gerada por esta versão do algoritmo é mostrada na Figura 3.

A representação escolhida foi a cartesiana pois permite visualização mais prática da trajetória do robô. Na figura, deve-se assumir que os lados esquerdo e direito estão conectados, uma vez que a figura é uma projeção imperfeita da superfície de uma esfera. Cada ponto amarelo é um ponto discretizado das linhas de solda. Os traços pretos representam a trajetória feita pelo robô. Apesar da distância percorrida de 613 m ter sido pequena, a trajetória final faz 18 fugas da trilha de solda, o que não é uma solução satisfatória para o problema.

A partir destes resultados, o algoritmo foi alterado objetivando melhorar o seu desempenho. Foi aplicada uma matriz de penalidades variável que permitisse às formigas passarem duas vezes pelo mesmo ponto, mas sem permitir caminhos na diagonal. Esta opção, porém, provou-se muito custosa computacionalmente e, apesar de possibilitar resultados melhores, torna o algoritmo muito pouco prático para aplicação a problemas reais.

A falha em obter uma solução aceitável do primeiro algoritmo mostrou a necessidade de alterar inteiramente o modo de encarar o problema, levando ao desenvolvimento do segundo algoritmo.

B. Segunda Abordagem

O segundo algoritmo desenvolvido foi criado aprendendo a partir dos erros do primeiro algoritmo. Pelos seus resultados, ficou claro que o método de otimização por colônia de formigas funciona para minimizar a rota, mas a fuga das trilhas de solda tornavam o algoritmo inaplicável. A penalidade aplicada nas trajetórias das formigas que se movem na diagonal minimiza a quantidade de fugas das trilhas de solda mas não é o suficiente para gerar resultados sem fugas. Concluiu-se que o método de representação por discretização era o problema, pois exige um número muito alto de pontos e, consequentemente, possui um número muito alto de possíveis resultados, levando a busca de outro método de representar o problema.

Sabendo que as trilhas de solda são retas sobre uma esfera, decidiu-se por representar pares de pontos de intersecção entre

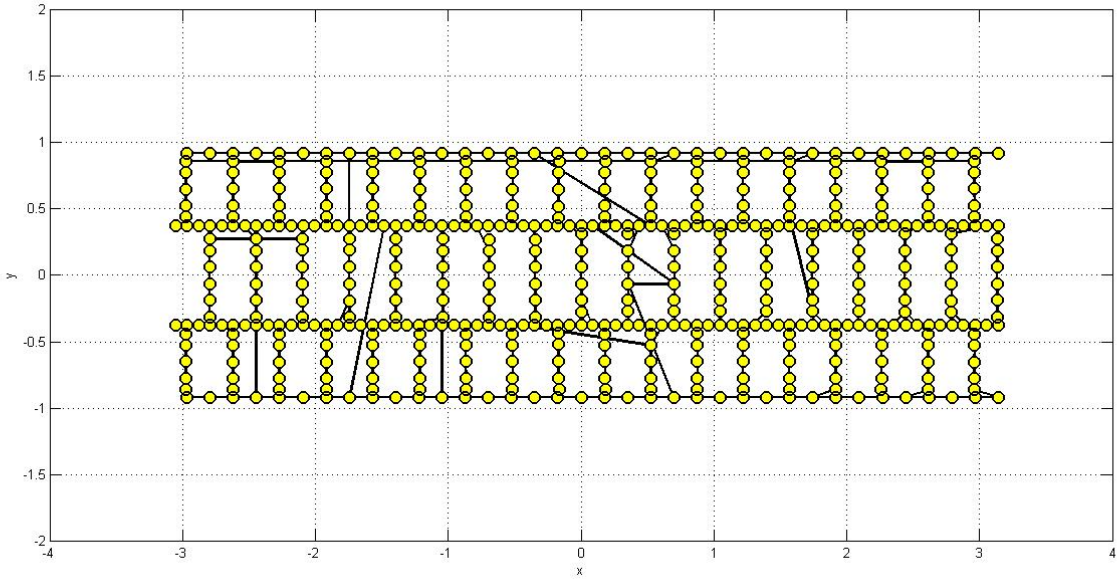

Figura 3: Melhor solução por abordagem de trilhas de solda discretizadas.

trilhas sem interrupção. Nesta nova concepção, as formigas deverão escolher um ponto, escolher uma trilha de solda que parta deste ponto, percorrer a trilha até o próximo ponto e repetir até que todas as trilhas de solda tenham sido percorridas. Como existe a garantia de cobrir todas as trilhas de solda, nenhuma matriz de penalidade é empregada, reduzindo o custo computacional.

A principal alteração nesta segunda abordagem foi o modo de execução do algoritmo. Ao invés de discretizar os cordões de solda, os únicos pontos utilizados são as intersecções entre duas trilhas, e uma nova matriz $2 \times k$, onde k é o número de trilhas de solda entre intersecções, foi criada para armazenar os pontos que possuem uma trilha de solda entre eles. Deste modo, o cálculo matricial tornou-se muito mais rápido uma vez que o número de pontos caiu de 504 para 108. Um pseudocódigo da segunda abordagem pode ser visualizado no Algoritmo 1. Neste novo algoritmo o número de iterações não se baseia mais na matriz de distâncias, mas na nova matriz proposta que representa as trilhas. Deste modo, as formigas são obrigadas a percorrer ao menos uma vez cada cordão de solda presente no tanque esférico.

Por causa da iteração em cima da matriz de trilhas, este novo algoritmo não elimina pontos já visitados da matriz de distâncias e permite ao robô andar até o mesmo ponto quantas vezes fossem necessárias, de modo a garantir que todas as trilhas fossem inspecionadas. A cada iteração, ao invés de procurar um novo ponto, as formigas escolhem um segmento de trilha de solda a ser inspecionado baseando-se no feromônio e na distância de ambos os pontos que definem a trilha, decidem qual dos dois pontos do segmento escolhido está mais perto da sua posição atual, andam até o ponto, percorrem o segmento de trilha de solda andando até o ponto em que ela termina e, em seguida, escolhem outro segmento para percorrer. A Figura 4 apresenta uma das melhores soluções geradas, sua trajetória total foi de 561 metros.

A solução foi considerada melhor que a abordagem anterior pois não realiza fugas das trilhas de solda, percorre uma distância total menor e, o que é mais importante, não deixa trilhas de solda sem inspecionar. As formigas dão preferência a trilhas ainda não percorridas que comecem no ponto onde já estão, permitindo, assim, garantir uma trajetória concisa. Esta preferência é passada a elas através da matriz de distâncias, que não possui diagonal principal igual a zero. Esta matriz é utilizada como peso para o cálculo do próximo ponto de cada formiga. Definindo o peso da distância de um ponto a ele mesmo com um valor alto, é possível permitir as formigas conectarem uma trilha na outra e, portanto, minimizar possíveis fugas.

Este algoritmo, apesar de simples, foi capaz de superar todas as outras tentativas em todos os quesitos: tempo de processamento, distância da menor trilha e quantidade de fugas de trilhas de solda. A seção de resultados dá avaliações numéricas para comparação.

IV. Resultados e Discussão

Para avaliar a performance dos métodos apresentados, foi implementado também um algoritmo MATLAB solucionador de problema de caixeiro viajante através de troca de laços (*loop flipping*), uma solução computacionalmente simples e rápida para o problema, disponível em [19]. A implementação foi feita em cima das trilhas de solda discretizadas. Nenhum algoritmo parecido com a segunda abordagem foi encontrado na literatura para comparação, devido a sua especificidade em abordar trilhas ao invés de pontos. Foram realizados

Algoritmo 1: Estratégia de otimização de trajeto de inspeção por colônia de formigas que garante cobertura total dos cordões de solda.

Input: Parâmetros do tanque esférico: centro, raio, e posição e orientação dos círculos que definem cordões de solda
Output: Menor caminho encontrado que inclua todos os cordões de solda

```
/* Carrega parâmetros e encontra intersecções das trilhas de solda */
pontos ← intersec_cordoes(center, r, cordoes[:])
n ← size(pontos, 1)
/* Passa pra coordenadas esféricas (raio é constante e pode ser ignorado) */
[azi, ele, r] ← cart2sph(pontos(:,1), pontos(:,2), pontos(:,3))
/* Cria matriz de distâncias D [n x n] com diagonal principal diferente de 0 para
   permitir que as formigas continuem no mesmo ponto */
D ← diag(10^-5)
for i ← 0 : 1 : n do
    for j ← i + 1 : 1 : n do
        [dist, ang] ← distance(ele(i), azi(i), ele(j), azi(j), 'radians');
        D(i, j) ← r. * dist
        D(j, i) ← D(i, j)
/* Define-se uma matriz T [k x 2] com as intersecções que formam trilhas de solda
   do tanque, onde k é o número de trilhas */
T ← trilhas(azi, ele, r)
/* Definem-se os parâmetros do algoritmo de otimização por colônia de formigas */
[MaxIt, nAnt, alpha, beta, tournament, rho] ← [35, 5, 0.5, 2, 4, 0.15]
/* Para cada iteração: */
for it ← 0 : 1 : MaxIt do
    /* Define-se a matriz de feromônios F [n x n] */
    F ← diag(10^-5)
    /* Para cada formiga: */
    for a ← 0 : 1 : nAnt do
        /* Para cada trilha de solda: */
        for t ← 0 : 1 : k do
            /* Peso de locomoção por F e D ponderados pelos parâmetros alfa e beta */
            i ← ant(k).Tour(end)
            P ← F(i,:)^alpha * D(i,:)^-beta
            /* Soma cumulativa dos pesos normalizados das trilhas */
            Q ← cumsum(normalize(P(T)))
            /* Por torneio, seleciona a próxima trilha com os pontos correspondentes */
            prox ← torneio(Q, tournament)
            ant(a).tour ← [ant(a).tourT(1, prox(1))model.T(2, prox(2))]
        /* Atualiza tour da formiga atual e o melhor tour */
        Best_Cost ← atualiza_custo(ant(a).tour, Best_Cost)
    /* Atualiza feromônios e evapora */
    F ← atualiza_feromonio(ant(a).tour, F, rho)
/* Mostra melhor solução */
exibe_resultado(Best_Cost)
```

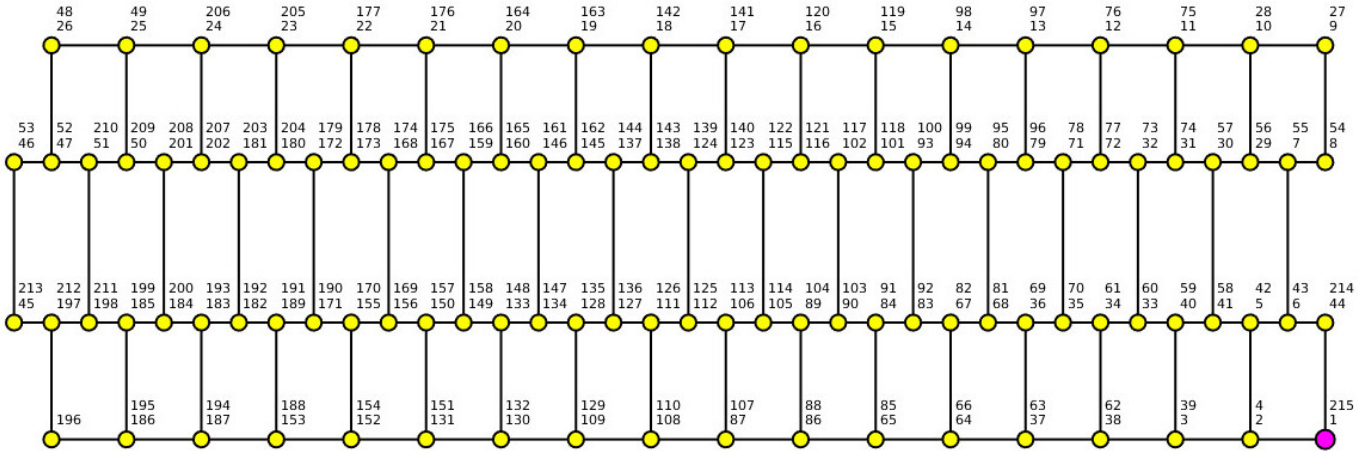

Figura 4: Melhor solução por abordagem de pontos de intersecção.

Algoritmo	Distância	Tempo
Primeira abordagem	590.34m	2m46s
	634.55m	3m01s
	630.27m	2m55s
Segunda abordagem	561.59m	2.6s
	585.82m	2.58s
	52.13m	2.57s
Loop Flip TSP Solver	600.67m	4.09s
	601.05m	4.25s
	598.11m	4.1s

Tabela I: Valores de tempo de execução e distância da melhor rota para três experimentos feitos com cada algoritmo testado no MATLAB.

três ensaios para comparar os valores de tempo de execução e distância da melhor rota. A Tabela I mostra os valores encontrados. Os valores de tempo de execução mostrados na tabela foram obtidos executando os algoritmos em um computador rodando Ubuntu 16.04, com hardware de 16GB de RAM e Intel i7-6700 3.4GHz. A versão do MATLAB utilizada foi a R2018a x64.

Como é possível observar nos resultados, a primeira abordagem é excessivamente demorada pois utiliza vários laços de execução e um número elevado de pontos devido ao uso de punição e à discretização. Além disso, há muita variação na distância da melhor rota e não há garantia de que o caminho inclua todas as trilhas de solda. A primeira versão apresentou muitas falhas que tornaram esta versão do algoritmo inadequada para a finalidade. Porém, a experiência foi essencial para o desenvolvimento do segundo algoritmo.

O algoritmo utilizado é bastante otimizado, evidente nos resultados de tempo de execução, e consegue encontrar rotas suficientemente eficientes, porém ainda com o problema de não cobrir todas as trilhas de solda, como é possível observar na figura 5 que mostra uma rota gerada por este algoritmo.

Como não é possível gerar uma trajetória inválida (que não percorra todas as trilhas) com a segunda abordagem, poucas iterações e poucas formigas foram necessárias para um resultado satisfatório. A média de convergência para uma trajetória com menos de 600 metros (tanque com raio de 9.125 metros) que passe por todas as trilhas é de 35 iterações utilizando 5 formigas. O gráfico da Figura 6 mostra a curva de *fitness* de uma rodada. A média de tempo de execução para estes valores é de 2.58 segundos, muito superior em relação à primeira versão do algoritmo, que levava em torno de 3 minutos para uma solução que poderia não ser satisfatória.

V. Conclusão

A inspeção completa de tanques de petróleo é de extrema importância para a prevenção de acidentes e desastres ambientais. Tal inspeção é realizada por robôs que devem, necessariamente, percorrer uma trajetória que cubra toda a extensão das trilhas de solda. A otimização por colônia de formigas é um método heurístico adequado para otimização de rotas e sua aplicação neste problema levou a resultados satisfatórios em distância da rota e velocidade de processamento. O estudo de caso realizado sugere fortemente a aplicabilidade do método para problemas reais e em escala muito maior. Futuramente, é possível incluir vários outros modelos de tanques, tanto esféricos quanto cilíndricos, de modo a aumentar sua aplicabilidade. Converter o algoritmo para linguagem C também deve apresentar um melhor desempenho em questão de tempo de processamento. Por fim, os objetivos foram considerados alcançados pela segunda abordagem do algoritmo.

Agradecimentos

N. Dalmedico e A.S. de Oliveira agradecem ao apoio financeiro da Agência Nacional do Petróleo, Gás Natural e Biocombustíveis (ANP), da Financiadora de Estudos e Projetos (FINEP), do Ministério da Ciência, Tecnologia, Inovações e Comunicações (MCTIC) por meio do Programa de Recursos Humanos da ANP para o Setor Petróleo e Gás (PRH-ANP), da CAPES e do Programa de Formação de Recursos Humanos da Petrobrás (PRH10-UTFPR). H.S. Lopes agradece ao CNPq pelas bolsas de pesquisa 311778/2016-0 e 423872/2016-8, bem como à Fundação Araucária pelo suporte financeiro através do PRONEX 042/2018.

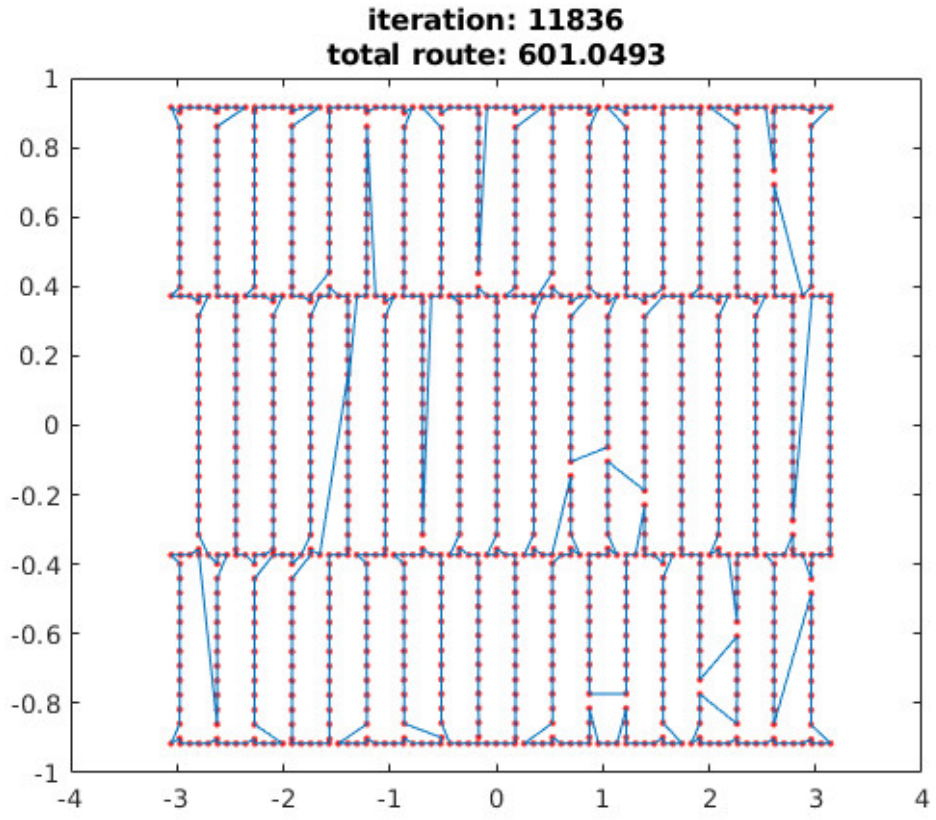

Figura 5: Solução encontrada pelo algoritmo solucionador de problema de caixeiro viajante em linhas de solda discretizadas.

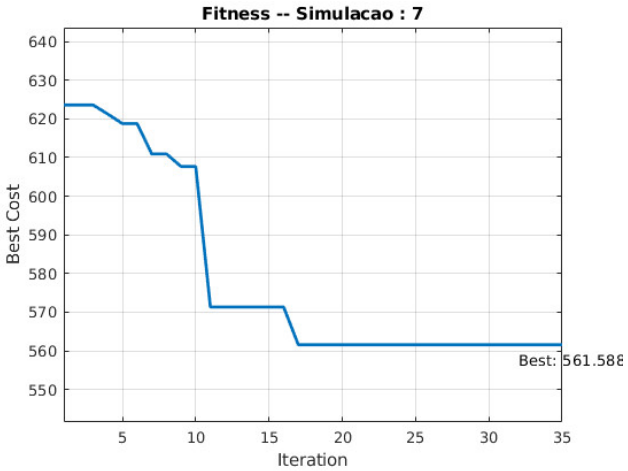

Figura 6: Gráfico de convergência (*fitness*).

REFERÊNCIAS

[1] Petronotícias. (2014) Petrobrás reajusta valor do gás liquefeito de petróleo. [Online]. Available: https://petronoticias.com.br/archives/62026

[2] R. S. da Veiga, A. S. de Oliveira, L. V. R. de Arruda, and F. Neves Junior, "Localization and navigation of a climbing robot inside a LPG spherical tank based on dual-lidar scanning of weld beads," in *Robot Operating System (ROS)*. Springer, 2016, pp. 161–184.

[3] M. A. S. Teixeira, H. B. Santos, A. S. De Oliveira, L. V. R. De Arruda, and F. Neves, "Environment identification and path planning for autonomous ndt inspection of spherical storage tanks," in *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*. IEEE, 2016, pp. 193–198.

[4] H. B. Santos *et al.*, "Controle inteligente de posição e velocidade para um robô escalador com rodas direcionáveis," Master's thesis, Universidade Tecnológica Federal do Paraná, 2016.

[5] H. S. Lopes, L. C. de Abreu Rodrigues, and M. T. A. Steiner, Eds., *Meta-Heurísticas em Pesquisa Operacional*, 1st ed. Curitiba, PR: Omnipax, 2013.

[6] A. Ismail, A. Sheta, and M. Al-Weshah, "A mobile robot path planning using genetic algorithm in static environment," *Journal of Computer Science*, vol. 4, no. 4, pp. 341–344, 2008.

[7] S. Mittal and K. Deb, "Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms," in *Proc. IEEE Congress on Evolutionary Computation*. IEEE, 2007, pp. 3195–3202.

[8] O. Castilho and L. Trujilo, "Multiple objective optimization genetic algorithms for path planning in autonomous mobile robots," *Soft Computing*, vol. 11, no. 3, pp. 269–279, 2007.

[9] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, USA: MIT Press, 2004.

[10] R. Silva, H. S. Lopes, and W. Godoy Jr., "A heuristic algorithm based on ant colony optimization for multi-objective routing in vehicle ad hoc networks," in *Proc. of BRICS Conference on Computational Intelligence*, 2013.

[11] Y. Li, H. Soleiman, and M. Zohal, "An improved ant colony optimization algorithm for the multi-depot green vehicle routing problem with multiple objectives," *Journal of Cleaner Production*, vol. 227, pp. 1161–1172, 2019.

[12] Y.-H. Huang, C. A.Blazquez, S.-H. Huang, G. Paredes-Belmar, and G. Latorre-Nuñez, "Solving the feeder vehicle routing problem using ant colony optimization," *Computers & Industrial Engineering*, vol. 127,

pp. 520–535, 2019.
[13] S. F. Ghannadpour and A. Zarrabi, "Multi-objective heterogeneous vehicle routing and scheduling problem with energy minimizing," *Swarm and Evolutionary Computation*, vol. 44, pp. 728–747, 2019.
[14] M. P. Garcia, O. Montiel, O. Castillo, R. Sepúlveda, and P. Melin, "Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation," *Applied Soft Computing*, vol. 9, no. 3, pp. 1102–1110, 2009.
[15] I. Châari, A. Koubaa, H. Bennaceur, S. Trigui, and K. Al-Shalfan, "Smartpath: a hybrid ACO-GA algorithm for robot path planning," in *Proc. IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–8.
[16] S. M. K. Heris, "Ant colony optimization for traveling salesman problem," 2015. [Online]. Available: http://yarpiz.com/53/ypea103-ant-colony-optimization
[17] D. Legland, "Geom3D library," 2015. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/24484-geom3d
[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, USA: Addison-Wesley, 1989.
[19] Y. Nativ. (2009) Mathworks file exchanger - another tsp solver. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/24857-another-tsp-solver