UNIVERSIDADE DE SÃO PAULO
Escola Politécnica

ALGORITMO DE PLANEJAMENTO DE TRAJETÓRIA PARA OBTENÇÃO DE ROTA DE FUGA

Natasha Mitie Tanoue

São Paulo
2013

Natasha Mitie Tanoue

ALGORITMO DE PLANEJAMENTO DE TRAJETÓRIA PARA OBTENÇÃO DE ROTA DE FUGA

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Bacharel em Engenharia Mecatrônica.

Área de concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Emílio Carlos Nelli Silva

São Paulo
2013

FICHA CATALOGRÁFICA

Tanoue, Natasha Mitie
Algoritmo de planejamento de trajetória para obtenção de rota de fuga / N.M. Tanoue. -- São Paulo, 2013.
55 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Otimização topológica 2.Algoritmos 3.Método dos elementos finitos I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

AGRADECIMENTOS

Agradeço ao meu melhor amigo e companheiro João Pedro por toda paciência e apoio durante os momentos difíceis na elaboração deste trabalho. Obrigada pelo carinho, força e incentivo dados para que isto fosse possível. Com certeza tudo seria muito mais difícil sem você ao meu lado.

Agradeço também ao meu professor orientador Prof. Dr. Emílio Carlos Nelli Silva por toda dedicação e conhecimento transmitido, fornecendo toda orientação necessária para facilitar a elaboração desta difícil tarefa.

RESUMO

A obtenção de rotas de fuga por meio de programação consiste na solução de um problema de planejamento de trajetória. Esse tópico, extensamente abordado, possui como solução típica a representação do espaço utilizando grafos e o uso de algoritmos de busca para sua resolução.

Recentemente, foi apresentado um novo método para resolução do problema de planejamento de trajetória utilizando otimização topológica. Esse método consiste em aplicar otimização topológica em uma analogia de condução de fluxo de calor para encontrar a trajetória ótima entre dois pontos previamente determinados.

Neste trabalho, utiliza-se esse método para desenvolver um algoritmo cuja função é encontrar rotas de fuga para pessoas presas dentro de edificações em situações de emergência. Para o desenvolvimento, empregam-se conceitos de otimização topológica e método de elementos finitos aliados à compreensão de fatores que possivelmente influenciam a mobilidade de pessoas em situações de risco.

Palavras-chave: Otimização topológica, Condução de fluxo de calor, Planejamento de trajetória, Rota de fuga dentro de edifícios, Algoritmo de rota de fuga.

ABSTRACT

Obtaining escape routes via software programming consists of the resolution of a path planning problem. This topic, extensively addressed, has as a typical solution the representation of the space by graphs and the usage of search algorithms to solve them.

Recently, a new method is presented for the resolution of the path planning problem via topology optimization. This method consists in applying the topology optimization in an analogy of heat flow conduction theory, in order to produce the optimal trajectory between two previously set points.

In the present study, the method just introduced is used in the development of a computer algorithm to find escape routes for people trapped inside buildings. This algorithm employs topology optimization concepts along with the finite elements method taking factors that may influence the crowd mobility in emergency situations into consideration.

Keywords: Topology optimization, Heat flow conduction, Path planning, Evacuation route inside buildings, Evacuation algorithm

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

A obtenção de rotas de fuga dentro de edificações em situações de emergência é um problema de resolução complicada, já que nestas situações, o estado do ambiente geralmente não é estático. Paredes e tetos que sofreram grandes danos podem desabar, bloqueando possíveis rotas de fuga ou ferindo pessoas; a ocorrência de incêndios e a consequente geração de fumaça tóxica inviabilizam a passagem por determinadas áreas do ambiente e apresentam um enorme risco à vida; além disso, o pânico gerado nesse tipo de situação pode prejudicar ainda mais o discernimento das vítimas. Dessa forma, existem diversos fatores que devem ser levados em consideração para resolução do problema de obtenção de uma rota de fuga ótima.

Segundo Pu e Zlatanova (2005), para a elaboração de um algoritmo de planejamento de rota de fuga eficiente, é necessário considerar as mudanças que ocorrem no sistema com o passar do tempo. Os principais fatores que devem ser considerados são os fatores ambientais (estado de danos, toxicidade, estado da energia e capacidade das rotas) e humanos (densidade populacional, idade e gênero, nível de deficiência e efeitos do terreno). Estes fatores influenciam tanto a escolha de rota de fuga segura a ser percorrida, quanto a mobilidade das vitimas na movimentação dentro da edificação.

O problema de obtenção de rota de fuga pode ser tratado como um problema de planejamento de trajetória, sendo que a abordagem geralmente adotada para sua solução é a representação do espaço por meio de um grafo, estrutura composta por vértices e arestas, e a utilização de um algoritmo computacional para encontrar a melhor trajetória por meio dele.

Recentemente, um novo método para solução de problemas de planejamento de trajetória foi proposto por Ryu et al. (2012) e utiliza otimização topológica aplicada a uma analogia de condução de fluxo de calor. Essa abordagem necessita que os pontos inicial e final da trajetória sejam fornecidos, o que é uma desvantagem em relação à utilização de grafos, que necessita apenas do ponto inicial; porém, o novo método permite que a trajetória ótima seja encontrada diretamente, já que não realiza busca por todos os caminhos possíveis e apresenta menor custo computacional para planejamento de grandes trajetórias ou alto refinamento da malha (Andrien, 2012). Além disso, possibilita a definição de diferentes mobilidades para o ambiente (Ryu et al., 2012), o que não é facilmente aplicável à estrutura de grafos.

O objetivo deste trabalho é desenvolver um algoritmo que encontre a melhor rota de fuga dentro de edificações em situações de emergência considerando mudanças nas condições ambientais e diferentes mobilidades no terreno, baseando-se no algoritmo que implementa o método desenvolvido por Ryu et al.. Este algoritmo será implementado, simulado e testado no software *MATLAB*, com o intuito de verificar sua eficiência e obtenção de conclusões.

Na seção 2.1 é feito um estudo do estado da arte sobre o problema apresentado, na seção 2.2 apresenta-se a fundamentação teórica. O método de otimização topológica para resolver problemas de planejamento de trajetória é apresentado na seção 2.3. Na seção 2.4 temos a implementação numérica e na seção 2.5 são apresentados os resultados obtidos. Na seção 2.6 é apresentada a comparação entre o método de otimização topológica e do algoritmo A*. Na seção 3, faz-se uma breve discussão do trabalho e a conclusão é, enfim, apresentada na seção 4.

1.1. Motivação

Na ocorrência de desastres naturais ou qualquer tipo de situação de emergência, decisões sobre quais rotas tomar e qual sentido seguir são cruciais para a sobrevivência de pessoas presas dentro de edificações. Uma decisão errada pode causar a perda de incontáveis vidas, como no caso do incêndio da boate Kiss em Santa Maria em 2013 (Folha UOL, 2013).

É extremamente complicado para as pessoas que se encontram em situações de emergência conseguir pensar de forma clara e objetiva a fim de fazer as melhores escolhas sobre como agir em um momento desses e, além disso, as mesmas podem não estar familiarizadas com a arquitetura do lugar (Shi e Zlatanova, 2005), o que gera maior dificuldade na escolha de uma rota de fuga. Por esses motivos, um dispositivo que auxilie na escolha do melhor caminho a ser percorrido para minimizar o tempo de fuga se faz necessário.

Esse dispositivo forneceria a trajetória ótima de fuga e se comunicaria com o sistema de monitoramento da edificação. Esse sistema seria formado por vários tipos de sensores presentes em diversos locais do prédio, com o objetivo de comunicar ao dispositivo qualquer alteração ocorrida no ambiente, como uma passagem bloqueada ou a presença de fumaça tóxica. A partir dessas informações, o dispositivo atualizaria a rota fornecida, fazendo as alterações necessárias para garantir que o usuário percorra o caminho em segurança. Poderia ser utilizado tanto por alguém preso no prédio, como por um bombeiro que realiza o resgate das vítimas.

Assim, a elaboração de um algoritmo que possa ser utilizado nesse dispositivo deve fornecer o melhor caminho a ser percorrido no menor tempo possível e deve levar em consideração as mudanças que ocorrem no ambiente para garantir a segurança de quem percorre o trajeto indicado.

O objetivo deste trabalho é desenvolver um algoritmo que apresente essas características e possibilite a evacuação de pessoas presas dentro de edificações em situações de emergência.

2. DESENVOLVIMENTO

2.1. Estudo do Estado da Arte

O problema de planejamento de trajetória para obtenção de rota de fuga é um assunto que foi extensamente abordado, porém os métodos utilizados para sua resolução apresentam poucas diferenças entre si.

Yoon (2012) utiliza e compara três algoritmos - de Dijkstra, de Johnson e o algoritmo de otimização de colônia de formigas (*ant colony optmization* – ACO) - para obtenção de uma rota de fuga em tempo real a partir do grafo que representa a faculdade de *Parkland*. Comparando os algoritmos de Dijkstra e de Johnson, observa-se que ambos obtêm a mesma rota, porém o algoritmo de Johnson forneceu a rota em um tempo menor. Já o ACO consegue encontrar rotas em pouco tempo e, ainda, mostra-se útil após novas barreiras serem adicionadas à trajetória original; contudo possui um alto custo computacional em comparação com o algoritmo de Johnson. Dessa forma, sugere-se utilizar o ACO modificado, em que inicialmente se utiliza o algoritmo de Johnson para encontrar uma rota inicial, então se inicializa o nível de feromônio com essa rota e atualiza-se a rota de fuga utilizando o ACO.

Yuan e Wang (2007), também utilizam o algoritmo de otimização de colônia de formigas para encontrar uma rota de fuga em situações de emergência, descritas por meio de um grafo, visando minimizar o tempo de fuga e a complexidade do trajeto a ser percorrido. Esses dois objetivos são combinados de forma a compor uma função objetivo única a ser minimizada. Também fornecem algumas informações sobre fenômenos que ocorrem em situações de fuga, como a variação da velocidade das vítimas de acordo com o tempo e com o trajeto que percorrem e a necessidade em se obter uma trajetória de fuga simples, que possa ser percorrida mesmo por pessoas desorientadas e em pânico.

Já Wei et al. (2012) combinam a teoria de autômatos celulares com o algoritmo de otimização de colônia de formigas para a obtenção de uma trajetória ótima de fuga em caso de situações de emergência. O ambiente proposto é composto por uma região de perigo, que representa a região do acidente, que pode ser, por exemplo, um incêndio, e por um ou mais pontos de escape, que são os pontos para os quais as vítimas tentarão se dirigir. Esse ambiente é modelado por meio de um grafo de duas dimensões. A matriz de decisão que irá movimentar cada uma das vítimas

através do ambiente, até um dos pontos de saída, leva em consideração dois fatores principais: o comportamento individualista do ser humano, que o fará buscar a rota mais curta para o escape, e o comportamento coletivista humano, que o fará considerar a movimentação das demais pessoas no momento de sua própria movimentação. Notou-se que a fuga simulada ocorreu de forma parecida com uma fuga real e não se deu pela menor trajetória possível, tendo sido influenciada pela natureza coletiva do ser humano.

Rajagopalan et al. (2007) utilizam o algoritmo de planejamento hierárquico (*hierarchical path planning algorithm* - HPPA) para encontrar rotas de fuga em tempo real, o qual tem por objetivo reduzir amplamente o espaço de busca de rotas por meio da utilização de um grafo abstrato no qual a cada um dos nós se associa um valor estimado de risco, de forma que a melhor trajetória será aquela com menor acúmulo de risco. Realizam um estudo da complexidade computacional do algoritmo quando destinado a situações estáticas e dinâmicas e comparam com outros dois algoritmos de Dijkstra (algoritmo da trajetória mais curta e algoritmo com poda, ou *pruning*), concluindo que o algoritmo proposto possui menor complexidade computacional.

Uma abordagem interessante é a sugerida por Bhushan et al. (2012). Nesse trabalho é apresentado um algoritmo para posicionamento de escadas em localidades ótimas, como janelas e varandas, baseado na distribuição de pessoas dentro do prédio. Dessa forma, são geradas mais opções de fuga e o tempo médio de evacuação ou o tempo de saída (*evacuation egress time*) é minimizado. Utiliza o modelo de grafos, onde associa a cada nó sua capacidade máxima, ou seja, número máximo de pessoas que podem ser acomodadas. Uma escada pode não aguentar uma carga igual ao tempo de viagem multiplicado pela sua capacidade de admissão, dessa forma, introduz-se um parâmetro denominado máxima carga que corresponde ao numero máximo de pessoas que podem estar na escada ao mesmo tempo. Através da realização de simulações, chega-se à conclusão que seu desempenho é equiparável ao de algoritmos existentes.

No método de Ryu et al. (2012) o objetivo é encontrar a melhor trajetória a ser percorrida por um robô móvel pontual em um ambiente cheio de obstáculos. Os obstáculos são modelados como isolantes enquanto as áreas livres são modeladas como condutores térmicos. Os pontos inicial e final são definidos como uma fonte e um sumidouro de calor, respectivamente, e a trajetória percorrida pelo fluxo de calor é a trajetória ótima. Essa trajetória é obtida a partir da minimização da energia térmica do sistema.

A partir do estudo de estado da arte, foi possível observar que a maior parte dos trabalhos destinados à resolução de problemas de planejamento de trajetória faz uso da estrutura de grafo e encontra a melhor trajetória por meio de um algoritmo computacional.

De forma geral, essa abordagem permite que a saída mais próxima seja encontrada fornecendo-se apenas o ponto inicial, o que é uma vantagem em relação à abordagem que será utilizada neste trabalho. Porém, para planejamento de grandes trajetórias ou alto refinamento da malha, esse tipo de abordagem apresenta um custo computacional maior do que o método de otimização topológica (Andrien, 2012). Além disso, o método utilizado neste trabalho possibilita que sejam incluídas diferentes mobilidades no decorrer de cada terreno, o que não é facilmente aplicável à estrutura de grafos.

Como essas duas abordagens são significativamente diferentes, não é possível aproveitar a maior parte das teorias fornecidas pelos artigos estudados. Utilizaram-se apenas conceitos referentes ao comportamento humano em situações de emergência e alterações na mobilidade devido a fatores dinâmicos.

2.2.Fundamentação teórica

O método de otimização topológica para planejamento de trajetória resolve este problema por meio de ferramentas de métodos de elementos finitos. Nesta seção, a fundamentação teórica de otimização topológica e do método de elementos finitos para transferência de calor são apresentadas.

2.2.1. Métodos de Elementos Finitos para Transferência de Calor

O método de otimização topológica para solução de problemas de condução de calor utiliza ferramentas de elementos finitos para realização de cálculos.

Seja a equação de Poisson em duas dimensões e seja T a função temperatura:

$$\nabla^2 T + f(x,y) = 0 \tag{1}$$

Sendo a solução aproximada definida como $\tilde{T}$.

No Método de Resíduos Ponderados, o resíduo da solução aproximada é dado por:

$$R = \frac{d^2\tilde{T}}{dx^2} + \frac{d^2\tilde{T}}{dy^2} + f(\tilde{x}, \tilde{y}) \quad (2)$$

e deve-se encontrar uma solução aproximada que minimize o resíduo de acordo com a fórmula:

$$\iint_D RW_i dD = 0\,, i = 1,2, \dots, m \quad (3)$$

onde D é o domínio da solução e W_i são as funções de peso linearmente independentes.

A abordagem mais comum é o emprego da função de forma N_i como a função peso, a qual é chamada de Método de Galerkin. Assim, neste caso tem-se:

$$\iint_D RN_i dD = 0\,, i = 1,2, \dots, m \quad (4)$$

Ao aplicar o Método de Resíduos Ponderados juntamente com o Método de Galerkin para solução da equação:

$$\nabla^2\tilde{T} + f(x, y) = 0 \quad (5)$$

Obtém-se

$$\iint_{A_e} N_i R dA = 0\,,\ i = 1,2, \dots, m\ (m \text{ é o } número\ total\ de\ nós) \quad (6)$$

$$\iint_{A_e} N_i(\nabla^2\tilde{T} + f(x, y))dA = 0,\ i = 1,2, \dots, m \quad (7)$$

$$\iint_{A_e}(\nabla^2\tilde{T})dA = -\iint_{A_e} N_i f(x, y)dA \quad (8)$$

Aplicando o Teorema de Green (integração por partes)

$$\iint_S u\nabla^2 v dS = \oint_C u(\nabla v)\cdot\bar{n}dl - \iint_S(\nabla u)^T\cdot(\nabla v)dS \quad (9)$$

Obtém-se

$$\iint_{A_e} N_i \nabla^2 \tilde{T} dA = \oint_{C_e} N_i (\widetilde{\nabla T}) \cdot \bar{n} dl - \iint_{A_e} (\nabla N_i)^T \cdot (\nabla \tilde{T}) dA \quad (10)$$

O primeiro termo do lado direito corresponde a uma integral de linha sobre o contorno de cada elemento, sendo que

$$\nabla \tilde{T} \cdot \bar{n} = \frac{\partial \tilde{T}}{\partial n} \quad (11)$$

$$\tilde{T} = \sum_{j=1}^{4} N_j T_j \Rightarrow \nabla \tilde{T} \cdot \bar{n} = \sum_{j=1}^{4} (\nabla N_j) \cdot \vec{n} T_j \quad (12)$$

Este termo fornece a condição de contorno natural do problema (condição de Neumann, ou derivada normal). Do segundo termo do lado direito da Eq. 10 resulta a matriz de rigidez do elemento.

O vetor de carregamento é dado por:

$$\iint_{A_e} f(x,y) N_i dA \quad (13)$$

Utilizando o elemento isoparamétrico de quatro nós como elemento de discretização, mostrado na fig. 1,

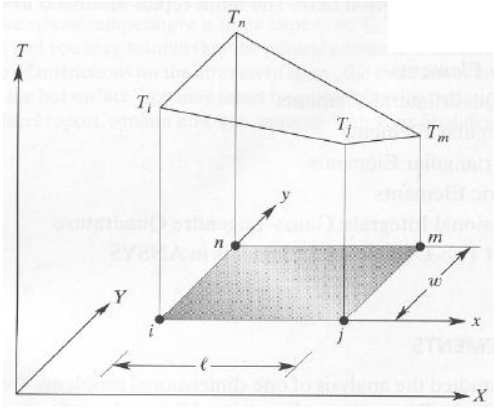

Figura 1- Representação de elemento isoparamétrico de 4 nós (retirada de Silva, apostila de Mecânica Computacional)

Considera-se a função interpoladora para um elemento retangular:

$$T(x,y) = b_1 + b_2 x + b_3 y + b_4 xy \quad (14)$$

sendo T(x,y) a variável de estado, b_i os coeficientes e x e y as variáveis independentes.

Considerando os valores dessa função T nos nós do retângulo tem-se:

$$T_i=b_1 \tag{15}$$

$$T_j=b_1+b_2L \tag{16}$$

$$T_m=b_1+b_2L+b_3W+b_4LW \tag{17}$$

$$T_n=b_1+b_3W \tag{18}$$

Resolvendo para os b_i e substituindo novamente em T(x,y):

$$T(x,y)=\left[\left(1-\frac{x}{L}\right)\left(1-\frac{y}{W}\right) \quad \frac{x}{L}\left(1-\frac{y}{W}\right) \quad \frac{xy}{LW} \quad \frac{y}{W}\left(1-\frac{x}{L}\right)\right]\begin{Bmatrix}T_i\\T_j\\T_m\\T_n\end{Bmatrix}=\begin{bmatrix}N_i & N_j & N_m & N_n\end{bmatrix}\begin{Bmatrix}T_i\\T_j\\T_m\\T_n\end{Bmatrix} \tag{19}$$

Assim obteve-se a temperatura escrita em função das temperaturas nodais do elemento. Para um problema de transferência de calor, tem-se as seguintes funções de forma:

$$\begin{cases}N_i=\left(1-\frac{x}{L}\right)\left(1-\frac{y}{W}\right)\\ N_j=\frac{x}{L}\left(1-\frac{y}{W}\right)\\ N_m=\frac{xy}{LW}\\ N_n=\frac{y}{W}\left(1-\frac{x}{L}\right)\end{cases}\Rightarrow\begin{cases}\frac{\partial N_i}{\partial x}=\frac{1}{LW}(-W+y)\\ \frac{\partial N_j}{\partial x}=\frac{1}{LW}(W-y)\\ \frac{\partial N_m}{\partial x}=\frac{y}{LW}\\ \frac{\partial N_n}{\partial x}=\frac{-y}{LW}\end{cases}\Rightarrow\begin{cases}\frac{\partial N_i}{\partial y}=\frac{1}{LW}(-L+y)\\ \frac{\partial N_j}{\partial y}=\frac{-x}{LW}\\ \frac{\partial N_m}{\partial y}=\frac{x}{LW}\\ \frac{\partial N_n}{\partial y}=\frac{(L-x)}{LW}\end{cases} \tag{20}$$

Definindo o vetor:

$$\boldsymbol{N}=\begin{Bmatrix}N_i\\N_j\\N_m\\N_n\end{Bmatrix} \tag{21}$$

Portanto,

$$\tilde{T}=\boldsymbol{N^T T}=\begin{bmatrix}N_i & N_j & N_m & N_n\end{bmatrix}\begin{Bmatrix}T_i\\T_j\\T_m\\T_n\end{Bmatrix} \tag{22}$$

e considerando a formulação obtida para a solução em cada elemento da Eq. 10 pela aplicação do Método de Resíduos Ponderados e Galerkin, tem-se:

$$\iint_{A_e} k(\nabla \boldsymbol{N})^T(\nabla \tilde{T})dA = \iint_{A_e} k\left\{\frac{\partial\{\boldsymbol{N}\}}{\partial x} \quad \frac{\partial\{\boldsymbol{N}\}}{\partial y}\right\}\begin{Bmatrix}\frac{\partial \tilde{T}}{\partial x}\\ \frac{\partial \tilde{T}}{\partial y}\end{Bmatrix}dA = \iint_{A_e} k\left\{\frac{\partial\{\boldsymbol{N}\}}{\partial x} \quad \frac{\partial\{\boldsymbol{N}\}}{\partial y}\right\}\begin{Bmatrix}\frac{\partial\{\boldsymbol{N}\}^T}{\partial x}\\ \frac{\partial\{\boldsymbol{N}\}^T}{\partial y}\end{Bmatrix}\boldsymbol{T}dA_e \tag{23}$$

Substituindo os valores das funções de forma, obtém-se:

$$= \iint_A \frac{k}{(LW)^2}\left\{\begin{Bmatrix}-W+y\\ W-y\\ y\\ -y\end{Bmatrix}\{-W+y \quad W+y \quad y \quad -y\} + \begin{Bmatrix}-L+x\\ -x\\ x\\ L-x\end{Bmatrix}\{-L+x \quad -x \quad x \quad L-x\}\right\}\boldsymbol{T}dA \tag{24}$$

Por fim, colocando *W*, *L* e *k* em evidência, obtém-se:

$$=\left\{\frac{kW}{6L}\begin{bmatrix}2 & -2 & -1 & 1\\ -2 & 2 & 1 & -1\\ -1 & 1 & 2 & -2\\ 1 & -1 & -2 & 2\end{bmatrix} + \frac{kL}{6W}\begin{bmatrix}2 & 1 & -1 & -2\\ 1 & 2 & -2 & -1\\ -1 & -2 & 2 & 1\\ -2 & -1 & 1 & 2\end{bmatrix}\right\}\boldsymbol{T} = \boldsymbol{K_e T} \tag{25}$$

Para facilitar os cálculos foi realizada uma simplificação do elemento de discretização escolhendo-se um elemento quadrado, L igual a W, sendo ambos unitários, assim como k. Dessa forma, a matriz $[K_e]$ fica reduzida à seguinte forma:

$$\boldsymbol{K_e} = \begin{bmatrix}2/3 & -1/6 & -1/3 & -1/6\\ -1/6 & 2/3 & -1/6 & -1/3\\ -1/3 & -1/6 & 2/3 & -1/6\\ -1/6 & -1/3 & -1/6 & 2/3\end{bmatrix} \tag{26}$$

Esta matriz de condutividade térmica é, por sua vez, utilizada nos cálculos do algoritmo de otimização topológica para problemas de condução de calor e para planejamento de trajetória.

Escrevendo-se as equações de forma sucinta, tem-se:

$$\boldsymbol{KT=F} \qquad \boldsymbol{K=}\overset{N}{\underset{1}{A}}\boldsymbol{K_e}\ ; \qquad \boldsymbol{F=}\overset{N}{\underset{1}{A}}\boldsymbol{F_e} \tag{27}$$

2.2.2. Otimização Topológica

A Otimização Topológica consiste num método computacional que permite projetar a topologia ótima de estruturas a partir da distribuição de material dentro de um domínio fixo, seguindo um certo critério de custo (por exemplo, máxima rigidez e para um certo peso). O material em cada ponto do domínio pode variar de vazio (não há presença de material) até sólido (total presença de material) podendo assumir densidades intermediárias entre vazio e sólido de acordo com um modelo de material definido.

Um problema que pode ser tratado por este método é o problema de condução de calor. A formulação típica para minimização de energia térmica dada por:

$$min\, \Pi = \boldsymbol{F^T U} = \boldsymbol{U^T K U} \tag{28}$$

Considerando o material a ser distribuído como isotrópico, utiliza-se o método do material solido isotrópico com penalização (*SIMP*, na sigla em inglês) para encontrar uma solução "0-1", em que 0 representa a área sem material e 1 a área preenchida. Neste método, as propriedades do material são supostas constantes em cada elemento de discretização do domínio de projeto e as variáveis são as densidades relativas do elemento. As propriedades do material são modeladas como o produto da densidade relativa do material elevada a um expoente pelas propriedades do material sólido. A função de custo é então dada por:

$$c(x) = \boldsymbol{U^T K U} = \sum_{e=1}^{N} (\boldsymbol{x_e})^p \boldsymbol{u_e}^T \boldsymbol{k_e u_e} \tag{29}$$

onde u_e é o vetor de temperatura e k_e é a matriz de "rigidez térmica" de cada elemento, x é o vetor de variáveis de projeto, N é o número de elementos da discretização e p é o expoente de penalização. Para o método SIMP, soluções adequadas são obtidas com $p \geq 3$.

As restrições deste problema são:

$$\frac{V(x)}{V_0} = f \tag{30}$$

$$0 \leq x_{min} \leq x \leq 1 \tag{31}$$

onde *V(x)* é o volume de material, V_0 é o volume do domínio de projeto, *f* é a fração de volume e x_{min} é um vetor para evitar singularidades. O problema da Eq. 30 é resolvido por meio da utilização dos seguintes critérios de optimabilidade:

$$x_e{}^{novo} = \begin{cases} \max(0, x_e - m) & se\ x_e B_e{}^{\eta} \leq \max(0, x_e - m) \\ \min(1, x_e + m) & se\ x_e B_e{}^{\eta} \leq \min(1, x_e - m) \\ x_e B_e{}^{\eta} & caso\ contrário \end{cases} \tag{32}$$

onde m é um limite positivo de movimentação, η=1/2 é um coeficiente numérico de amortecimento e B_eé obtido a partir da condição de optimabilidade:

$$B_e = \frac{-\frac{\partial c}{\partial x_e}}{\lambda \frac{\partial V}{\partial x_e}} \tag{33}$$

Em que o multiplicador de Lagrange λ deve ser escolhido de forma que a restrição de volume seja satisfeita. O valor adequado pode ser encontrado por meio de um algoritmo de bipartição.

As sensibilidades da função objetivo *c* e do volume do material V com respeito ao elemento de densidades x_e são dadas por:

$$\frac{\partial c}{\partial x_e} = -p x_e{}^{p-1} u_e{}^T k_0 u_e \tag{34}$$

Para garantir a existência de soluções para o problema de otimização topológica e evitar a formação de padrões de "tabuleiro de xadrez", algumas restrições devem ser impostas no projeto.

Uma abordagem comum é a aplicação de filtros nas sensibilidades. O filtro de sensibilidade modifica as sensibilidades $\frac{\partial c}{\partial x_e}$ da seguinte forma:

$$\frac{\widehat{\partial y}}{\partial x_e} = \frac{1}{x_e \sum_{f=1}^{N} \widehat{H_f}} \sum_{f=1}^{N} \widehat{H_f} \frac{\partial c}{\partial x_f} \tag{35}$$

Sendo que H_f é um fator de peso definido como:

$$\widehat{H_f} = \left(r_{min} - dist(e, f) \right) \tag{36}$$

$$\{f \in N | \, dist(e,f) \leq r_{min}\}, \quad e = 1, \ldots, N \tag{37}$$

em que *dist(e,f)* é definido como a distância entre o centro do elemento *e* e o centro do elemento *f*. O operador de convolução $\widehat{H_f}$ vale zero fora da área do filtro e decai linearmente com a distância do elemento *f*. Ao invés das sensibilidades originais (Eq. 35) utiliza-se as sensibilidades modificadas (Eq. 36) no critério de optimalidade.

Ao resolver este problema de minimização de energia, gera-se uma estrutura que maximiza a transferência de calor por condução a partir da maximização da "rigidez térmica".

Um exemplo de aplicação deste método para problemas de condução de fluxo de calor é apresentado por Bendsoe e Sigmund (2003). Este código (anexo A) foi estudado e simulado, pois foi utilizado na implementação do método de otimização topológica para problemas de planejamento de trajetória e, assim, é de grande importância para posterior compreensão do código que implementa o método a ser utilizado neste trabalho.

O problema padrão a ser resolvido é mostrado na Fig. 2.

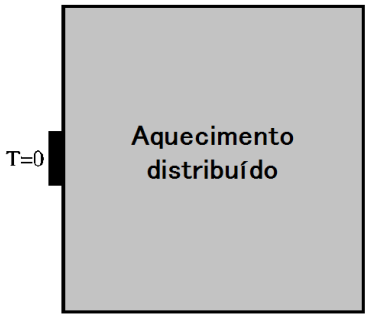

Figura 2- Problema padrão

Considera-se a distribuição de duas fases de materiais com condutividades isotrópicas iguais a 1 e 0,001 respectivamente. A placa quadrada é uniformemente aquecida e o centro da borda esquerda é um sumidouro de calor, ou seja, nesta região da placa, a temperatura é definida como sendo igual a zero.

A função utilizada é: `toph(nelx,nely,volfrac,penal,rmin)`. Em que:

- nelx: número de elementos na direção horizontal (x);
- nely: número de elementos na direção vertical (y);
- volfrac : valor de volume permitido para a estrutura final;

- penal: fator de penalização utilizado pelo método *SIMP* na atualização das variáveis;
- rmin: valor utilizado na aplicação do filtro.

Utilizando a chamada da função igual a `toph(40,40,0.4,3,1.2)` obtemos o seguinte resultado:

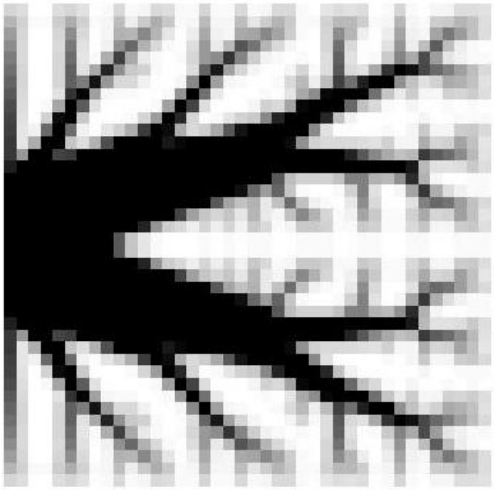

Figura 3 – Resultado da simulação de *toph(40,40,0.4,3,1.2)*

Para melhor compreensão dos resultados obtidos, foram realizadas mais simulações e os resultados comparados. Alguns deles são apresentados nas figuras 4 a 7.

Nas figuras 4 e 5 variou-se apenas o valor do parâmetro *rmin* em relação à fig. 3.

Na fig. 4 *rmin*=1 o que significa que o filtro está desativado. Por esse motivo, podemos observar na figura áreas em que há a presença de padrões quadriculados, chamado "efeito de tabuleiro de xadrez" comumente presente em problemas de otimização topológica. Esses padrões não são desejados, pois implicam em soluções impossíveis de serem aplicadas. Assim, a utilização de um filtro se faz necessária para evitar esse tipo de situação.

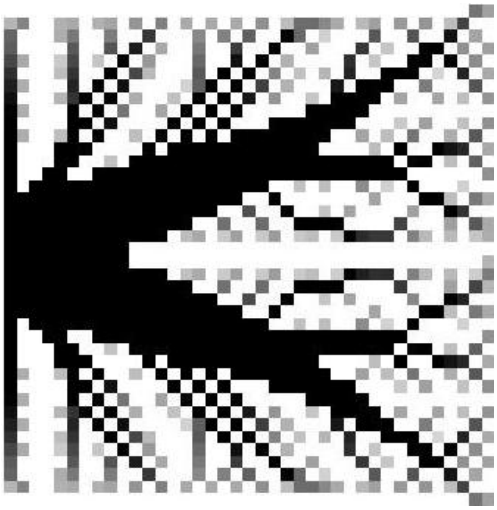

Figura 4 – Resultado da simulação de *toph(40,40,0.4,3,1)*

Já no caso da figura 5 *rmin*=1.5, um valor maior do que o utilizado na fig. 2. É possível notar que esta figura tem um aspecto mais "embaçado" do que a primeira.

Com o aumento do valor de *rmin*, uma quantidade maior de elementos foi utilizada para realizar a filtragem dos valores, dessa forma, os ramos mais finos, que estão rodeados por áreas "vazias" (em branco), tiveram seus valores diminuídos resultando numa representação em tons de cinza mais claro.

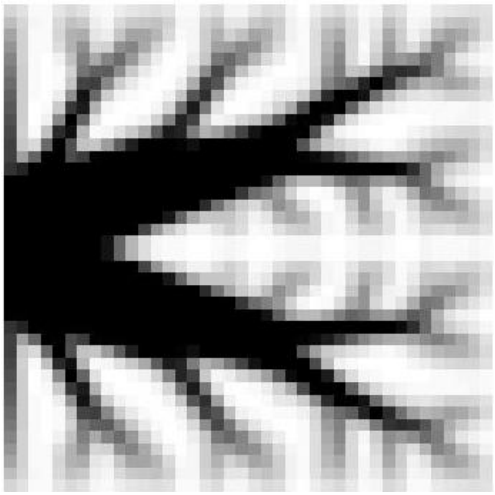

Figura 5 – Resultado da simulação de *toph(40,40,0.4,3,1.5)*

Como esperado, na figura 6 observa-se que o aumento do volume permitido implica numa porcentagem maior de massa distribuída pelo domínio.

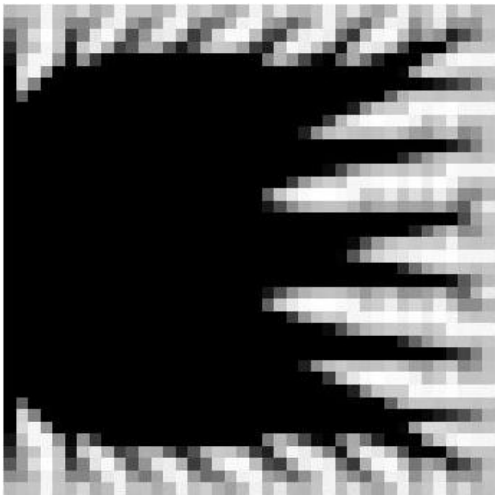

Figura 6 – Resultado da simulação de *toph(40,40,0.7,3,1.2)*

Por fim, na fig. 7 aumentou-se o refinamento da malha, utilizando 80X80 elementos eum volume permitido de 50%.

Graças ao maior número de elementos, o resultado final apresenta um maior número de ramificações, as quais estão mais bem distribuídas pelo material em comparação com os casos anteriores.

Esse resultado era esperado, já que um maior refinamento da malha implica na obtenção de soluções com uma resolução melhor, o que neste caso representa a possibilidade de ramos mais finos na distribuição do material.

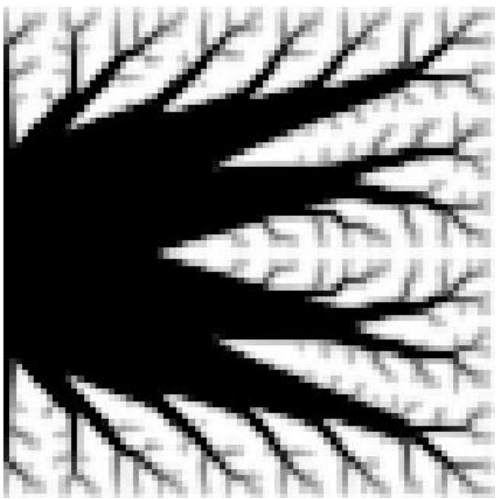

Figura 7 – Resultado da simulação de *toph(80,80,0.5,3,1.2)*

A partir deste estudo foi possível observar a grande influência que o filtro e o refinamento da malha exercem sobre os resultados. Por isso, merecem atenção especial quando da definição dos valores dos parâmetros a serem utilizados na chamada da função, de modo a garantir a viabilidade da solução a ser obtida.

O estudo e simulação deste código foram realizados sem maiores problemas graças à revisão bibliográfica realizada no começo do ano.

2.3. Método de Otimização Topológica para planejamento de trajetória

Esse método foi o primeiro a utilizar otimização topológica para solucionar o problema de planejamento de trajetória. O critério utilizado foi o de minimização da energia térmica do sistema em consideração.

2.3.1. Formulação do problema

O problema a ser resolvido é dado pela equação:

$$\nabla(-k\nabla\theta) + Q = 0, \ em\ \Omega \tag{38}$$

em que k é o coeficiente de condutividade térmica e Q representa geração de calor e θ $(= T - T_\infty)$ é a temperatura.

Para formular um problema de otimização topológica a condutividade térmica k^edo *e-ésimo* elemento finito deve ser interpolada como função da variável de projeto elemento de densidade $\gamma_e (0 \leq \gamma_e \leq 1)$. Um possível polinômio interpolador é:

$$k^e = k_e {\gamma_e}^p \tag{39}$$

$$\begin{cases} \gamma_e \to 1 : \Omega_e \to elemento\ material \\ \gamma_e \to 0 : \Omega_e \to elemento\ vazio \end{cases} \tag{40}$$

$$p = \begin{cases} 1 & para\ n \leq c \\ min(3{,}1 + 0{,}1 \times (n\text{-}10)\ para\ n > nc & (número\ de\ iterações) \end{cases} \tag{41}$$

Em que nc pode ser ajustado dependendo do problema (resultados satisfatórios foram obtidos para $(5 \leq nc \leq 10)$.

A formulação típica para minimização de energia térmica é:

$$min \prod = \boldsymbol{F}^T \boldsymbol{\theta} = \boldsymbol{\theta}^T \boldsymbol{K} \boldsymbol{\theta} \tag{42}$$

com restrição de massa (interpretada como o comprimento médio da trajetória)

$$\sum_{e=1}^{N_e} \gamma_e m_e - M_o(n) \leq 0 \tag{43}$$

$$0 < \varepsilon < \gamma_e \leq 1 (e = 1,2,...,N_e : \varepsilon = valor\ pequeno) \tag{44}$$

Em que:

$$M_0(n) = max(N(n), M_{min}) \tag{45}$$

$$N(n) = \frac{\sum_{e=1}^{N_e} me}{n^2} \tag{46}$$

$$M_{min} = \frac{\left|P_g - P_s\right| me}{\sum_{e=1}^{N_e} me} \tag{47}$$

Sendo P_g e P_s os vetores de posição do ponto de chegada e de partida, respectivamente, e $\left|P_g - P_s\right|$ a distância euclidiana entre eles.

A massa M_0 é reduzida continuamente pelo número de iterações n. A massa mínima $M_{mín}$ é definida como a distância euclidiana entre o ponto de partida e o de chegada. Dessa forma, nenhuma trajetória pode ser menor do que uma linha reta conectando os dois pontos.

No problema anterior, o calor era transferido por uma superfície e neste caso, o fluxo será concentrado, pois há apenas um ponto de sumidouro. Dessa forma, ambos maximizam a "rigidez térmica" da superfície, porém, neste caso, essa maximização tem como objetivo tornar o caminho percorrido pelo fluxo de calor o menor possível.

Como será discutido na próxima seção, existem diversos fatores que influenciam na mobilidade do terreno, e este modelo permite representar os diferentes níveis de mobilidade em diferentes áreas do terreno.

2.4. Implementação

O problema a ser tratado neste trabalho é o problema de obtenção de rotas de fuga para pessoas presas dentro de edificações em situações de emergência. Este problema consiste em um problema de planejamento de trajetória e é resolvido utilizando o método de otimização topológica.

2.4.1. Simulação do algoritmo de Van den Goor

Para um caso real, a determinação da posição atual seria feita a partir de um localizador, uma espécie de GPS para interiores de prédios, que poderia estar integrado ao dispositivo que mostra a trajetória. Já as condições do edifício seriam monitoradas por sensores dispostos de forma adequada para detectar qualquer mudança significativa no ambiente. Estes se comunicariam diretamente com o dispositivo, fazendo a atualização das condições de cada área em um determinado período de tempo, possibilitando a identificação das áreas seguras a serem utilizadas na composição da trajetória de fuga. Neste trabalho, as posições inicial e final, assim como modificações no ambiente são inseridas manualmente no *software* para fins de simulação.

Inicialmente, utilizou-se o algoritmo de Van den Goor para testar ideias concebidas no início do trabalho. Primeiramente, foi verificada a capacidade deste método de encontrar trajetórias à medida que as condições ambientais eram modificadas.

A planta da edificação é uma variável de entrada deste método e é representada por uma matriz quadrada, cujas células podem receber números de 1 a 4. Células com valor igual a 1 representam os obstáculos e com valor igual a 2 representam o espaço livre. O ponto inicial da trajetória é definido pela célula com valor igual a 3 e o ponto final, pela célula com valor igual a 4. Nesta primeira etapa, foram elaboradas várias matrizes para realização dos testes e as modificações nas condições ambientais foram realizadas manualmente, sendo os novos obstáculos introduzidos a cada trajetória calculada.

Após esta etapa, testou-se a ideia da representação de plantas de edificações com vários andares pela concatenação das matrizes que representam cada andar, juntamente com a representação da distância a ser percorrida de um andar a outro, como por exemplo:

$$\begin{pmatrix} 2 & \cdots & 2 \\ \vdots & \ddots & \vdots \\ 2 & \cdots & 2 \end{pmatrix} \quad \begin{pmatrix} E & \cdots & E \\ \vdots & \ddots & \vdots \\ E & \cdots & E \end{pmatrix} \quad \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}$$

Andar 2 Escada Andar 1

$$\begin{pmatrix} 2 & \cdots & 2\,E & \cdots & E\,1 & \cdots & 1 \\ \vdots & \ddots & \vdots\,\vdots & \ddots & \vdots\,\vdots & \ddots & \vdots \\ 2 & \cdots & 2\,E & \cdots & E\,1 & \cdots & 1 \end{pmatrix}$$

Matriz concatenada

Esta ideia pôde ser testada, já que o método de otimização topológica possui baixo custo computacional para trajetórias longas (Andrien, 2012). Dessa forma, pode-se obter uma solução viável para uma planta com vários andares.

A partir dos testes iniciais foi possível obter maior familiaridade com o algoritmo de Van den Goor, e tomando-o como base, foi implementado o algoritmo deste trabalho.

2.4.2. Busca de saídas

O primeiro passo foi a implementação da busca de saídas. Como dito anteriormente, a planta é representada por uma matriz quadrada, cujas células recebem valores de 1 a 4. Nesta abordagem, o ponto de chegada já deve ser conhecido para que o algoritmo seja simulado.

Assim, a nova matriz implementada possui uma dimensão a mais, que caracteriza o "cenário" que ela representa. Por exemplo, uma planta com três saídas possui três cenários, dessa forma, cada cenário possui apenas uma saída e representa uma opção de trajetória para fuga. Ela é construída a partir de uma matriz geral, que não contem as saídas, e as mesmas são adicionadas posteriormente. A cada novo cenário é necessário excluir a saída anterior para que não haja erro na execução do programa. Um exemplo é apresentado abaixo:

```
%%definição da matriz%%
ex1(1:10,1:10,1)=2;
ex1(3:7,4,1)=1;
ex1(6,5:9,1)=1;
ex1(1,1,1)=5;

%%saida cenario 1%%
ex1(10,10,1)=6;
```

```
%%saida cenário 2%%
ex1(:, :, 2) = ex1(:, :, 1);
ex1(10,10,2)=2; %exclusão da saída S1
ex1(5,10,2)=6; %definição da nova saída
```

A matriz de três dimensões que contém todos os cenários é a variável de entrada *MAP_cenarios.* Essa variável inicializa a variável *caminhos* e o vetor *comprimento_caminhos*, que é um vetor "1 x número de cenários", preenchido com zeros.

Dentro do *loop*, a primeira etapa consiste na limpeza e inicialização de variáveis, já que estas serão utilizadas repetidamente até que as rotas ótimas sejam obtidas para cada cenário. Realiza-se então o cálculo de cada trajetória. Ao final da obtenção da rota ótima para cada cenário, o comprimento do caminho calculado é armazenado no vetor *comprimento_caminhos* e uma imagem é gerada para mostrar o caminho obtido.

Ao final de todas as iterações, verifica-se qual caminho possui menor comprimento e uma figura é impressa mostrando o valor de seu comprimento. Caso haja mais de uma trajetória com o comprimento mínimo, serão impressas imagens correspondentes a cada uma delas.

Este algoritmo também pode ser utilizado para realizar modificações nas condições ambientais, definindo-se a cada cenário uma nova configuração de obstáculos.

2.4.3. Inclusão da modificação de mobilidade

A última etapa de implementação consistiu na inclusão de fatores humanos e condições complexas de terreno que influenciam na mobilidade ao se percorrer a trajetória.

Primeiramente, modificou-se a definição de alguns valores, sendo que o ponto de partida passou a ser definido pelo valor 5 e o ponto de chegada pelo valor 6. Os valores 3 e 4 passaram a representar certos níveis de mobilidade em cada ponto da planta.

Em seguida, foi realizada a inclusão de diferentes mobilidades. Esta inclusão foi feita a partir da multiplicação da variável otimizada xnew por uma constante entre 0 e 1. Esses valores representam a porcentagem de mobilidade, que varia de 0 a 100%, variando-se assim a condutividade localmente. Dessa forma, nas áreas da planta que recebiam o valor igual a 3, tem-se:

```
for n=1:mapsize(1)
    for m=1:mapsize(2)
        if MAP(n,m)==3
```

```
            xnew(n,m)=0.5*xnew(n,m);
        end
    end
end
```

Sendo que nestas áreas, a mobilidade era reduzida em 50%.

É importante notar que esta inclusão é feita de forma simples e permite definir várias áreas no terreno com diferentes valores de mobilidade que variam de 0 a 100%.

2.5. Resultados

Nesta seção serão apresentados os resultados dos testes realizados a cada etapa do trabalho e a cada tópico será realizada uma breve discussão sobre a influência das modificações realizadas.

2.5.1. Simulação do algoritmo de Van den Goor

A linha de chamada da função do código de Van den Goor é:

```
[obj vol path] = top88r(nelx,nely,nc,rmin,fast,Mn,MAP)
```

Em que:

- nc: parâmetro ajustável para definir o valor do fator de penalização;
- fast: variável binária que influencia a velocidade de construção da figura (0=devagar, 1=rápido);
- Mn: valor da massa permitida para a trajetória;
- MAP: matriz que representa a planta do local em que se deseja encontrar a trajetória. Nessa matriz já se identificam os obstáculos, ponto de partida e o ponto final da trajetória.

Os demais parâmetros possuem os mesmos significados apresentados anteriormente.

Para fins de simulação foram realizadas pequenas alterações no código original, e as mesmas foram mantidas na elaboração do algoritmo deste trabalho.

Primeiramente, retirou-se o parâmetro *Mn* da linha de chamada, pois o mesmo não era utilizado em nenhum cálculo do algoritmo, apenas recebia uma atribuição de valor numa das linhas do programa.

Outra modificação realizada foi a mudança da linha *xPhys=round(xPhys)* pela linha *xPhys=round(xPhys+0.11*MAP)*. Essa mudança se fez necessária, pois dependendo do valor de filtro utilizado, a trajetória era encontrada, mas a figura gerada não continha a trajetória completa. Como a função *round* arredonda os valores para o inteiro mais próximo, todos o valores abaixo de 0,5 eram zerados e não apareciam como parte da trajetória na figura gerada. Nas fig. 8 e fig. 9 podemos observar esse efeito.

Ambas as figuras foram geradas a partir da chamada *top88r(40,40,5,1.2,1,map1)*, antes e depois da mudança respectivamente.

Nas imagens apresentadas, as partes em cinza representam os obstáculos, a linha na cor preta representa a trajetória encontrada e o espaço em branco representa o terreno livre. O ponto de partida (fonte de calor) está identificado com a letra F e o ponto final (sumidouro de calor) está representado pela letra S.

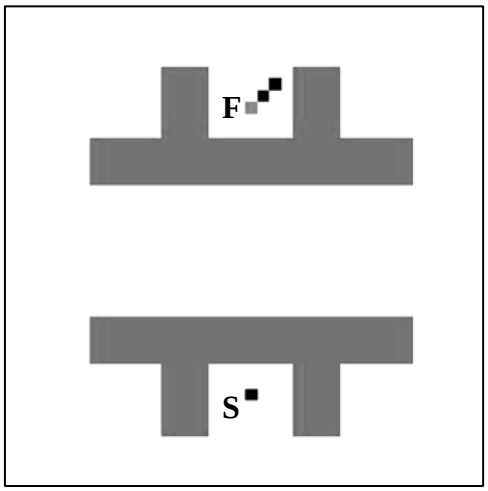

Figura 8 – resultado antes da modificação

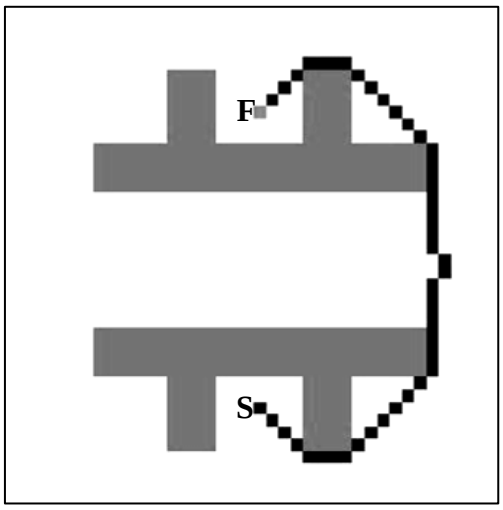

Figura 9 – Resultado após a modificação

Como é possível observar, sem a modificação a trajetória impressa fica incompleta, o que pode levar a conclusões equivocadas. (O valor de *0.11*MAP* foi arbitrado e, portanto, pode ser modificado caso haja necessidade).

2.5.1.1.Modificação nas condições ambientais

Para essa simulação foi construída uma matriz que representa um terreno com obstáculos, mostrada na fig. 10.

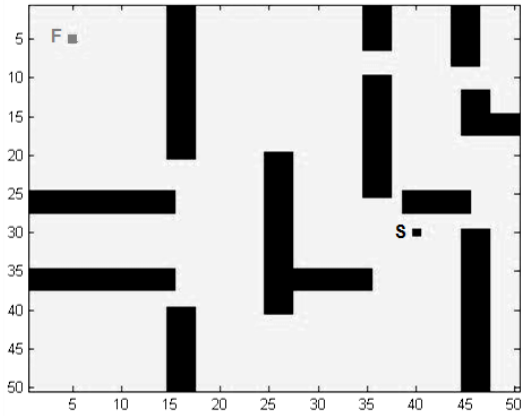

Figura 10 – Matriz para representação de uma planta

O resultado obtido para a chamada: *[obj vol path] = top88r(50,50,5,1.2,1,ex3)* é mostrado na fig. 11.

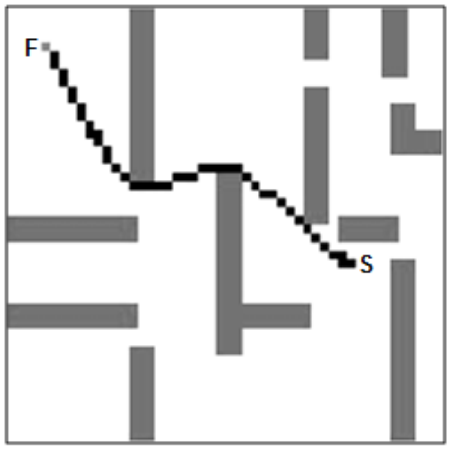

Figura 11 – Trajetória sem modificações

Nas figuras a seguir são apresentados resultados obtidos após modificações realizadas no ambiente mantendo-se os mesmos pontos inicial e final e a chamada *top88r(50,50,5,1.2,1,ex1).*

Figura 12 – Primeira mudança no ambiente

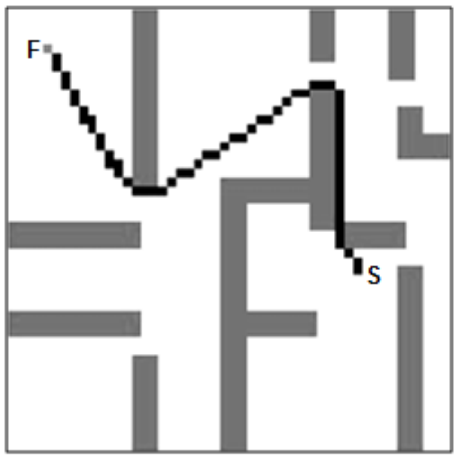

Figura 13 – Segunda mudança no ambiente (utilizou-se *0,14*MAP para a impressão)*

Das figuras apresentadas pode-se observar que a cada mudança ocorrida o *software* calcula uma nova trajetória a ser percorrida, permitindo verificar que este método é de fato capaz de modelar mudanças no ambiente.

É importante notar que neste código, também existe a influência da variação no valor das variáveis de entrada.

Na fig. 14 podemos observar o resultado obtido utilizando a chamada *top88r(50,50,5,1,1,ex1)* para uma planta em que os obstáculos são representadas por uma espessura menor.

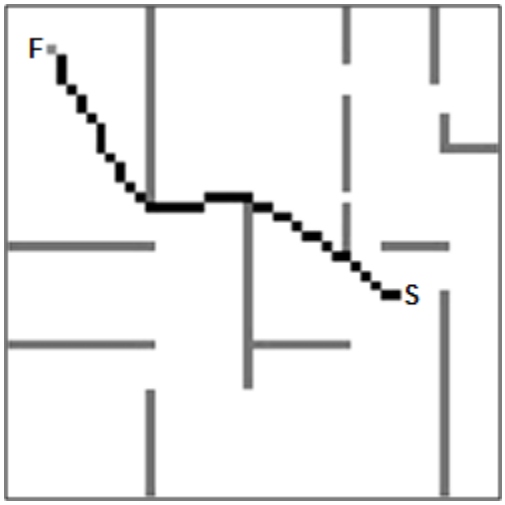

Figura 14 – Planta com menor espessura de obstáculo

Alterando o valor de *rmin*=1 para *rmin*=1.2 obtemos a seguinte figura:

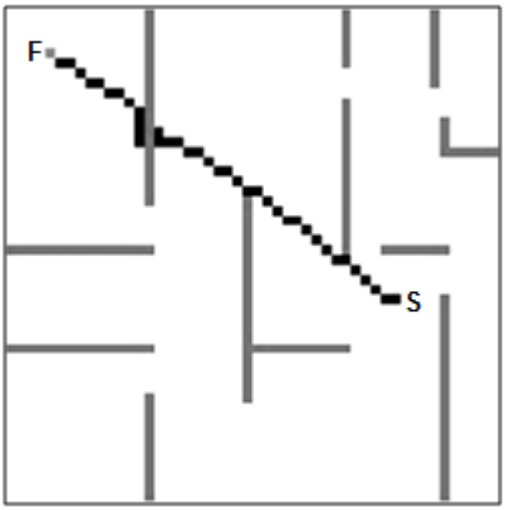

Figura 15 – Resultado da chamada *top88r(50,50,5,1.2,1,ex1)*

Como é possível observar na fig. 15, ao aumentar o valor do filtro, a trajetória calculada "atravessa" o obstáculo e, dessa forma, não pode ser utilizada como rota de fuga. Como explicado anteriormente, um valor maior de *rmin* aumenta o número de elementos utilizados na filtragem. Nesse caso, como o obstáculo é representado por uma linha simples, a maior parte dos valores ao redor dele é de células vazias (em branco), o que causou má interpretação do *software* e permitiu que uma rota fosse traçada através de um obstáculo.

Esse problema pode ser resolvido aumentando-se a espessura do obstáculo como na planta da fig. 11. Com um obstáculo de espessura maior, o valor de *rmin* exerce menor influência e permite a obtenção de uma trajetória ótima e viável.

2.5.1.2. Plantas com vários andares

A etapa seguinte do projeto é a representação e o cálculo de trajetórias para plantas com vários andares. Utilizando o terreno representado anteriormente, imaginou-se uma edificação composta de dois andares, com aquela distribuição de obstáculos, ligados por escadas. Assim, foi construída uma matriz de 110X110 a partir da concatenação de duas matrizes 50X50 mais uma matriz que representa as escadas que ligam um andar ao outro. Como a matriz dever ser sempre quadrada, as células ao redor da planta foram preenchidas com o valor que representa um obstáculo e as demais com um valor que representa um terreno livre. A fig. 16 apresenta a matriz explicada anteriormente.

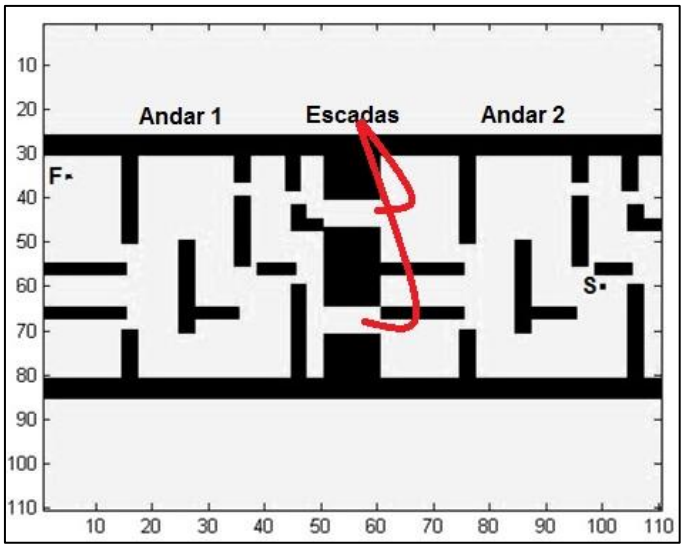

Figura 16 – Representação de dois andares

Para a planta de dois andares utilizou-se a chamada *[obj vol path]* = *top88r(110,110,5,1.2,1,pl1)* e obteve-se o seguinte resultado:

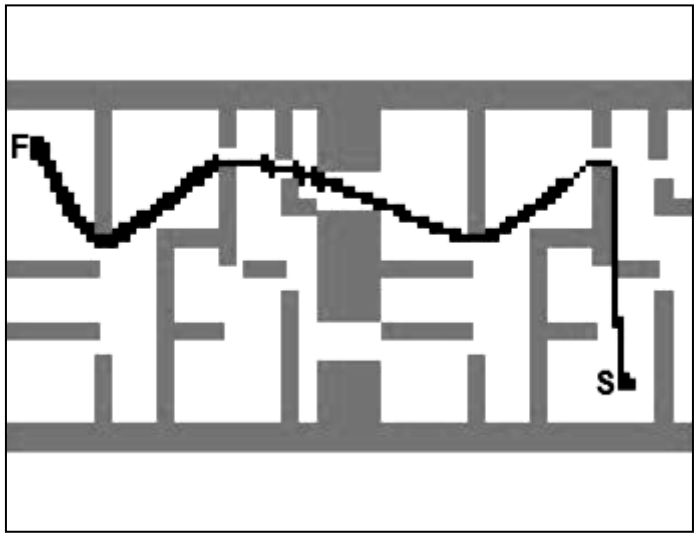

Figura 17 – Resultado para planta com dois andares (utilizou-se 0.22*MAP para impressão)

Já para uma planta com 4 andares dispostos em forma de "U" obteve-se o seguinte resultado:

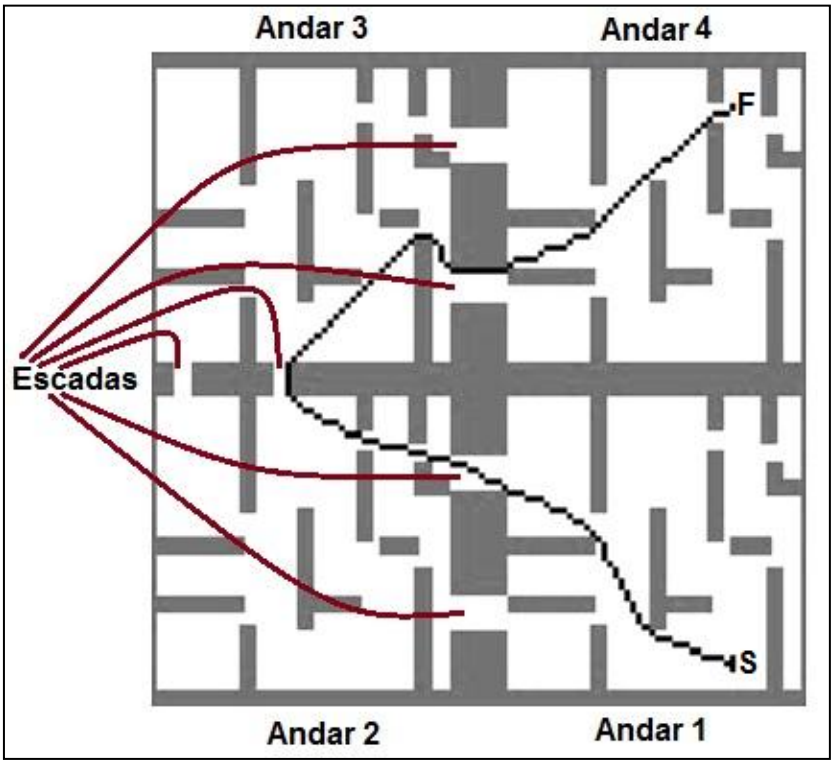

Figura 18 – Representação de 4 andares em "U" (utilizando 0.2*MAP para impressão)

Assim como no caso anterior, o algoritmo foi capaz de encontrar a trajetória entre os pontos fornecidos. Portanto verifica-se que a ideia sugerida de representação de uma edificação que contem vários andares (através da concatenação de matrizes que representam cada andar e as escadas que ligam um ao outro) é de fato viável e pode ser utilizada para o cálculo de trajetórias de rota de fuga.

2.5.2. Busca por saídas

Já foi observado que uma das desvantagens deste método é o fato de ser necessário definir os pontos inicial e final da trajetória para seu cálculo.

Utilizando a planta mostrada na fig. 11, foram definidas três saídas a serem consideradas pelo *software* mostradas na fig. 19.

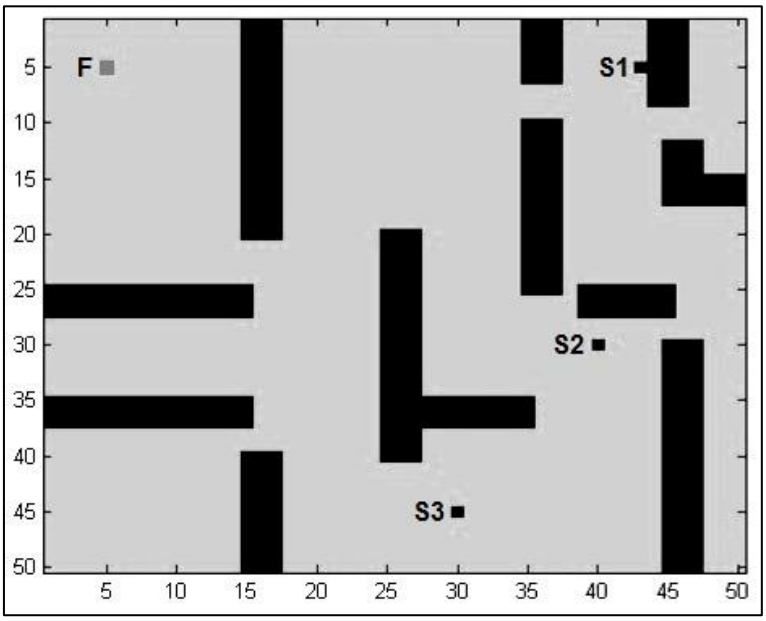

Figura 19 – Planta com múltiplas saídas

Utilizando o programa apresentado no apêndice A obtém-se os seguintes resultados:

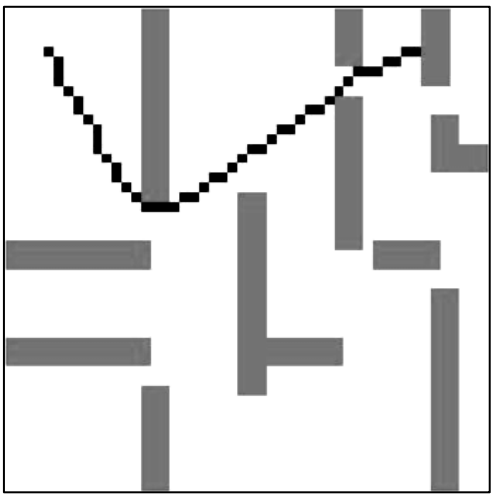

Figura 20 – Trajetória entre o ponto de partida até a primeira saída S1

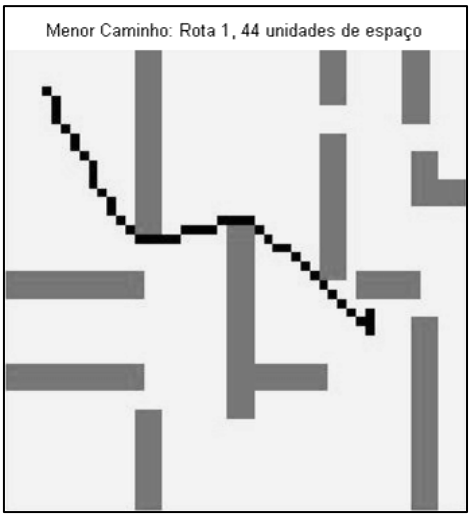

Figura 21 – Trajetória entre o ponto de partida até a segunda saída S2. Esta saída fornece uma trajetória de comprimento mínimo com 44 unidades.

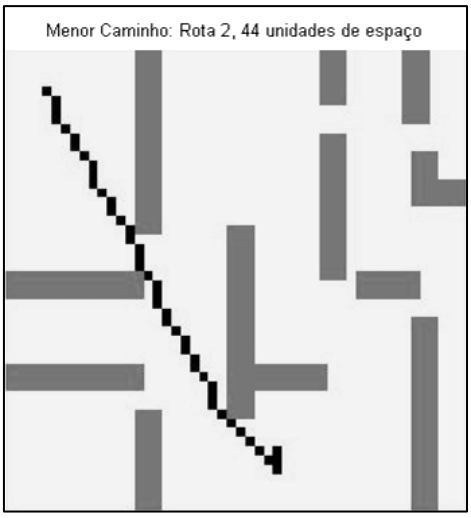

Figura 22 – Trajetória entre o ponto de partida até a terceira saída S3. Esta saída também fornece uma trajetória de comprimento mínimo com 44 unidades

Para fins de simulação, este programa imprime a trajetória calculada para cada saída e, ao final, imprime a trajetória ótima. Por coincidência, foram obtidas duas trajetórias com o mesmo comprimento, até S2 e S3, e o programa foi ajustado para imprimir todas as trajetórias de comprimento mínimo.

O tempo total para o cálculo das três trajetórias foi de aproximadamente 7,5s. É importante notar que para terrenos maiores e mais complexos, este tempo seria maior. Porém, para o dispositivo que utilizaria este programa, é razoável considerar que a atualização das condições ambientais seja realizada na escala de minutos. Dessa forma, esta pode ser uma solução para o para uma das desvantagens deste método, que é fato de ser necessário indicar os pontos inicial e final da trajetória.

2.5.3. Diferentes Mobilidades

Como discutido anteriormente, existem diversos fatores que influenciam na mobilidade de pessoas presas dentro edificações em situações de emergência. Nesta etapa, essa influência foi representada pela modificação do valor de mobilidade determinadas áreas do terreno, mostrada na fig. 23.

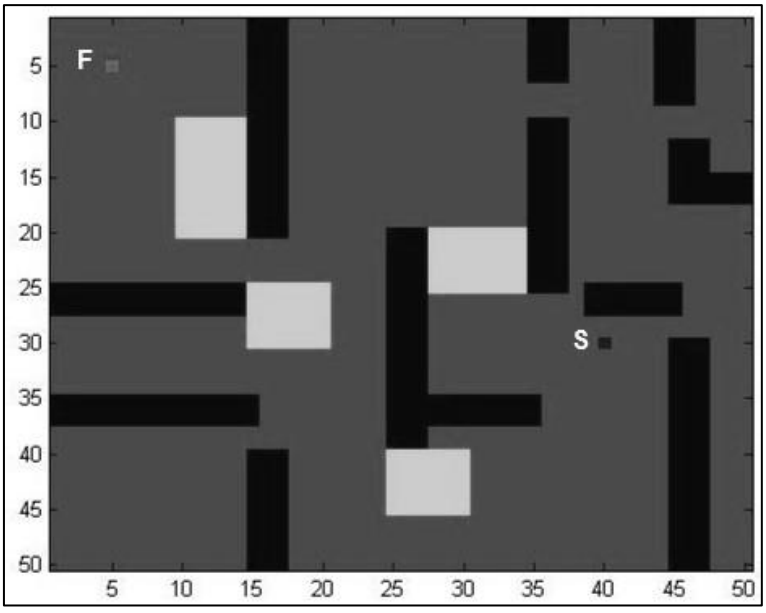

Figura 23 – Variação de mobilidade

Nesta figura, as áreas em cinza escuro são as áreas com mobilidade máxima e as áreas em cinza claro possuem mobilidade reduzida em 50%. As áreas em preto são os obstáculos.

Utilizando esta representação de terreno para o cálculo da trajetória, obtém-se:

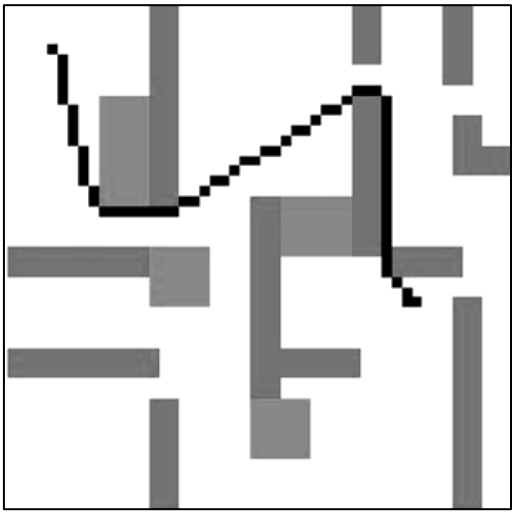

Figura 24 – Primeira modificação de mobilidade

Alterando novamente a mobilidade do terreno, obtém-se:

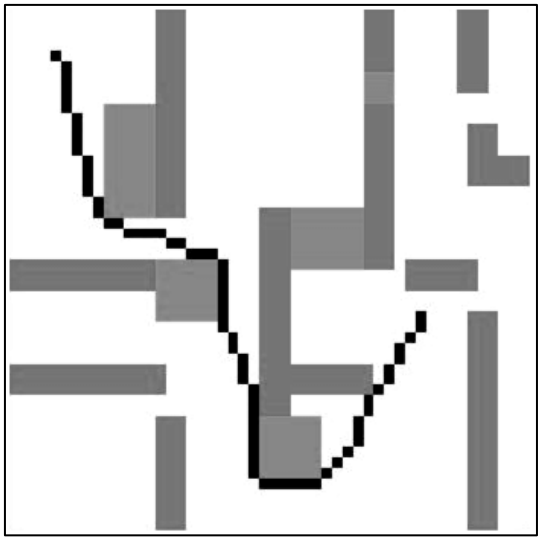

Figura 25 – Segunda modificação de mobilidade

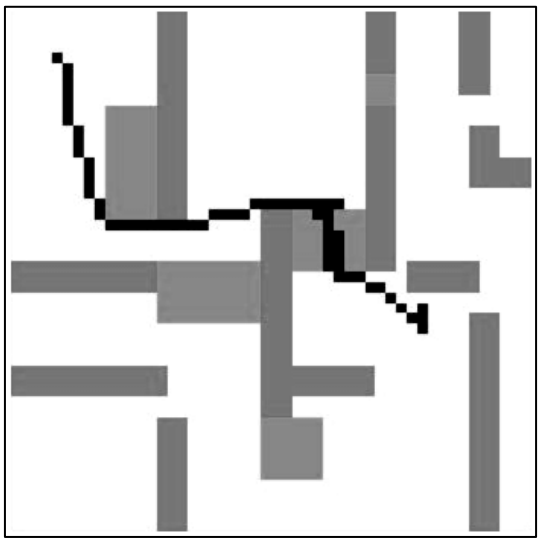

Figura 26 – Terceira modificação de mobilidade

Como é possível observar, ao calcular a trajetória, o algoritmo evita as áreas com menor mobilidade.

Esta é uma das grandes vantagens deste método, pois permite a inclusão de qualquer valor de mobilidade que varie de 0 a 100% em qualquer área do terreno, o que não é facilmente implementado em uma representação por grafos.

Ao possibilitar a inclusão desse fator na representação do terreno, é possível calcular rotas que não apenas são ótimas, mas que também maximizam a mobilidade do trajeto a ser percorrido. Dessa forma, o modelo do terreno e a trajetória calculada podem representar de forma bem próxima as condições reais da edificação, melhorando a eficiência do algoritmo.

Na próxima seção apresenta-se uma comparação deste método com o algoritmo de busca *A star (A*)*.

2.6. Comparação com o algoritmo A*

Como complemento ao trabalho, foi realizada a comparação deste método com o algoritmo de busca A*. O código utilizado para realização desta comparação é aquele apresentado por Andrien (2012) no apêndice C. Da mesma forma naquele trabalho, foram realizadas simulações em diversos cenários para verificar o desempenho de cada algoritmo e algumas delas são apresentadas a seguir.

As configurações do computador utilizado para esta comparação são:

- Sistema Operacional: Windows 7 - 64 Bits;
- Processador: Intel Core 2 Duo 2.13 GHz;
- Memória RAM: 4GB.

As comparações são apresentadas com o terreno utilizado, rota obtida por cada algoritmo e o tempo de processamento. Nas imagens a seguir, o lado esquerdo é sempre o resultado obtido pelo algoritmo A* e, o direito, aquele obtido pelo método de otimização topológica (MOT).

- **Caso 1:**

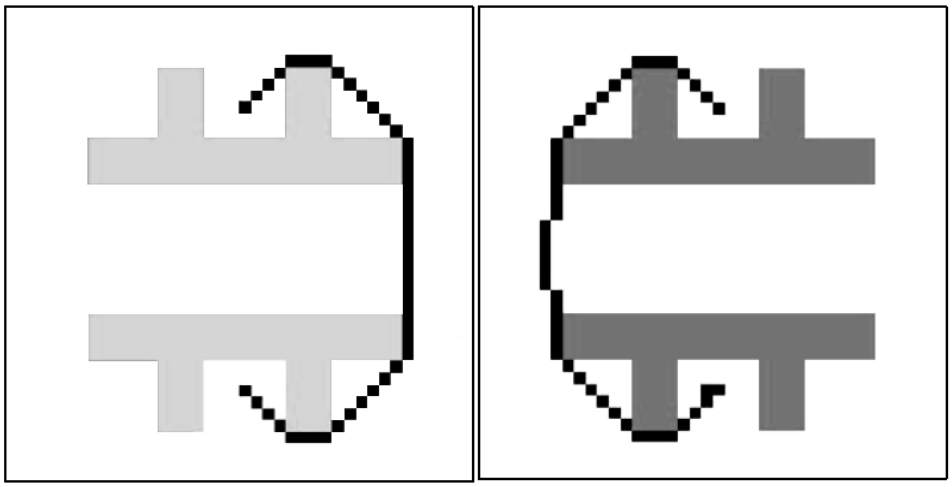

Figura 27 – resultado obtido ao realizar a primeira comparação entre os algoritmos. Lado esquerdo: A* - 2,5s. Lado direito: MOT – 1,959s

- **Caso 2**

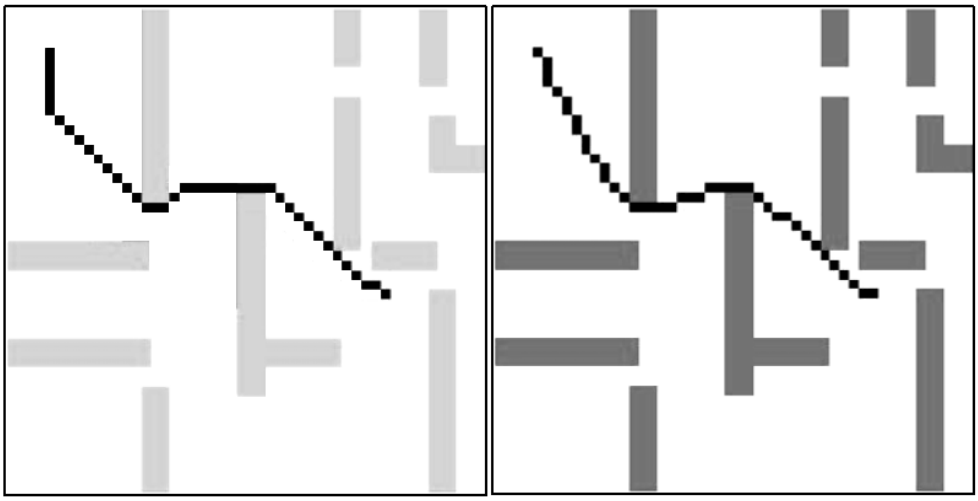

Figura 28 – resultado obtido ao realizar a segunda comparação entre os algoritmos. Lado esquerdo: A* - 0,757s. Lado direito: MOT – 2,809s

- **Caso 3**

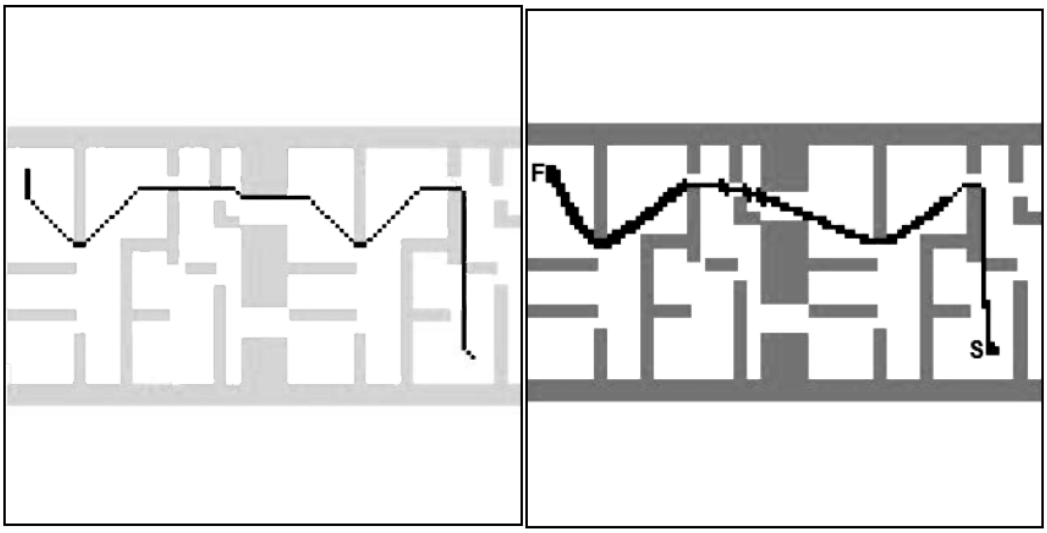

Figura 29 – resultado obtido ao realizar a terceira comparação entre os algoritmos. Lado esquerdo: A* - 6,816s. Lado direito: MOT – 11,365s

- **Caso 4**

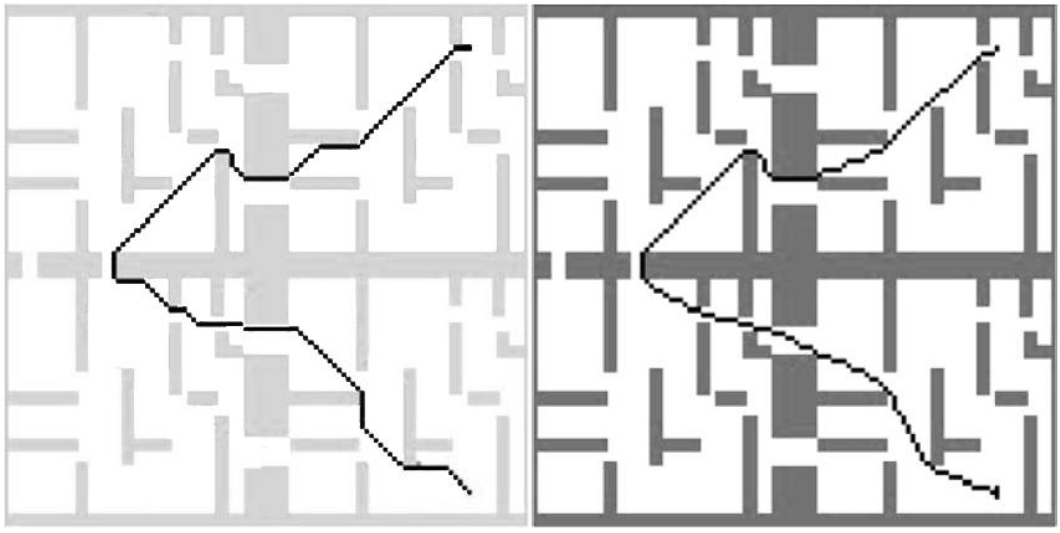

Figura 30 – resultado obtido ao realizar a quarta comparação entre os algoritmos. Lado esquerdo: A* 10,072s. Lado direito: MOT – 22.182s

Como é possível observar, o algoritmo A* é mais rápido na maioria dos casos. Porém, é importante ressaltar que para o dispositivo de busca de rotas, ambos apresentam resultados em tempos satisfatórios, já que é razoável considerar que a escala de tempo para atualização das informações no dispositivo seja de minutos.

Outro ponto é que, para todos os casos, ambos forneceram a mesma trajetória como resultado. Diferentemente do caso definido por Andrien (2012) em que a trajetória é ótima apenas se possuir o menor número de células preenchidas, para este problema de obtenção de rotas de fuga esta definição de trajetória ótima não se aplica. Para o robô, faz sentido estipular que num determinado corredor, a rota a ser percorrida seja a mais próxima da parede possível, porém para pessoas em situação de emergência, essa exigência perde a validade, pois o ser humano não segue instruções de forma rigorosa como um robô. Para o problema de rotas de fuga, o importante é determinar por qual corredor, porta ou sala que o usuário deve trafegar, de forma a garantir sua segurança no percurso. Assim, o método de otimização topológica é uma opção viável para planejamento de trajetória para obtenção de rotas de fuga.

Vale ressaltar, que sua maior vantagem é a possibilidade de definir de forma simples diversos valores de mobilidade para qualquer área do terreno, o que aproxima a solução da realidade e garante maior veracidade dos resultados.

3. DISCUSSÕES

Neste trabalho foi utilizado o método de otimização topológica para problemas de planejamento de trajetória desenvolvido por Ryu et al. (2012).

Por se tratar de um método recente, apresenta desafios a serem vencidos, como o cálculo apropriado do valor da massa M_0 para melhorar a robustez desse algoritmo (Andrien, 2012). Apesar disso, apresenta grandes vantagens como o baixo custo computacional para trajetórias longas (Andrien, 2012) e a possibilidade de incluir facilmente a influência de fatores humanos e ambientais na representação do espaço a ser percorrido.

A partir dos testes e simulações, foi possível observar que este método permite a representação de edificações com vários andares pela concatenação de matrizes e também possibilita a resolução do problema com múltiplas saídas, assim como os métodos que utilizam grafos. Além disso, ao compará-lo com o algoritmo A*, notou-se que também apresenta resultados satisfatórios para a obtenção de rotas de fuga, apesar de apresentar maior tempo computacional na maioria dos casos.

Devido à heurística do algoritmo A*, o custo dos nós numa diagonal direta são menores, dessa forma, estes são pesquisados primeiro. Por isso, nestes casos simulados ele apresenta um tempo computacional menor. Se a disposição dos obstáculos fosse diferente, de forma a não permitir a passagem pela diagonal, o tempo computacional aumentaria de forma significativa (Andrien, 2012). Já no caso do método de otimização topológica, o tempo computacional depende apenas do tamanho da planta de entrada, e não da disposição dos obstáculos. O que é uma vantagem em relação ao A*.

4. CONCLUSÕES

O objetivo deste trabalho foi o desenvolvimento de um algoritmo que encontre a melhor rota de fuga dentro de edificações em situações de emergência considerando mudanças nas condições ambientais e diferentes mobilidades no terreno, baseando-se no algoritmo que implementa o método de otimização topológica para planejamento de trajetória.

Como o algoritmo foi implementado em *MATLAB*, as mudanças nas condições ambientais e a busca por saídas foram implementadas de forma simples a partir da utilização de matrizes com três dimensões. Também de forma simples, a inclusão de diferentes mobilidades foi realizada a partir da multiplicação da variável otimizada por uma constante que varia de 0 a 1 representando mobilidades locais que variam de 0 a 100%.

A partir dos testes realizados, foi possível observar que o tempo computacional aumentava quanto maior fosse a matriz de entrada. Para o caso de uma matriz de 112X112 que representava uma edificação com quatro andares, o tempo computacional foi de aproximadamente 20s. Este é um tempo razoável para o cálculo da trajetória considerando que o tempo necessário para percorrer a trajetória por cada andar provavelmente será maior que o tempo computacional. Porém, se for necessário que o tempo computacional seja menor, é possível utilizar uma matriz que represente um andar, e calcular a trajetória para cada andar a medida que a trajetória seja percorrida.

Dos resultados obtidos, é possível concluir que este método é adequado para a obtenção de rotas de fuga e poderia ser utilizado em um dispositivo móvel para situações de emergência, pois possibilita a obtenção de rotas de fuga considerando os fatores citados anteriormente e possui um tempo computacional razoável para o cálculo da trajetória.

5. REFERÊNCIAS BIBLIOGRÁFICAS

ANDREASSEN, E. et al. Efficient topology optimization in MATLAB using 88 lines of code. *Structural and Multidisciplinary Optimization*,43(1), Springer-Verlag, 2010, p.1-16.

ANDRIËN, A. Topology Optimization versus A*: a comparison for 2D robot path planning. Bachelor Final Project, Departement of Mechanical Engineering, Eindhoven University of Technology, Springer-Verlag, 2012.

BENDSøE, M.P.; Sigmund, O. Topology optimization: theory, methods, and applications, Springer-Verlag, 2011, ISBN 3-540-42992-i.

BHUSHAN, A.; SARDA, N. L.; REDDY, P. V. R. Evacuation Planning of Large Buildings Using Ladders. *'DEXA (1)'*, India, Springer-Verlag, 2012, p.71-85.

FILIPPOUPOLITIS, A.; GELENBE, E. A distributed decision support system for Building Evacuation. *2009 2nd Conference on Human System Interactions*, Italy, 2009 p.323-330.

FOLHA UOL, 2013, *Tragédia no Sul*. Disponível em: <http://www1.folha.uol.com.br/especial/2013/tragediaemsantamaria/>. Acesso em Maio, 2013.

PU S.; ZLATANOVA S. Evacuation route calculation of inner buildings. *Geo-information for disaster management*, Netherlands, 2005, p.1143-1161.

RAJAGOPALAN, R. et al. Hierarchical path computation approach for large graphs. *IEEE Transactions onAerospace and Electronic Systems*, 44(2), 2008, p.427-440.

RYU, J.C.; PARK, F.C.; KIM, Y.Y. Mobile robot path planning algorithm by equivalent conduction heat flow topology optimization. *Structural and Multidisciplinary*, 45(5), South Korea, Springer-Verlag, 2012, p.703-715.

SIGMUND, O. A 99 line topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization*, Springer-Verlag, *21*(2), Denmark, 2001, p.120-127.

SILVA E.C.N. Apostila PMR2420 – Mecânica Computacional, São Paulo.

WEI, C. et al. A Research on Human Emergency Evacuation Based on Revised ACO-CA. *2012 20th International Conference on Geoinformatics*, China, 2012, p.1-7.

YOON, H. *Providing Optimized Evacuation Routes for Victims after a Disaster*. Structural Design Optimization Final Project, Department of Civil and Environmental Engineering, University of Illinois, Urbana-Champaign¸ USA, 2012.

YUAN, Y.; WANG, Q. Multi-Objective Path Selection Model and Algorithm for Emergency Evacuation. *IEEE International Conference on Automation and Logistics*, China, 2007, p.340-344.

ANEXO A

Código modificado para resolver problema de condução de calor (Bendsoe e Sigmund, 2003).

```
%%%% A 91 LINE MATLAB CODE FOR HEAT CONDUCTION PROBLEMS %%%
function toph(nelx,nely,volfrac,penal,rmin);
% INITIALIZE
x(1:nely,1:nelx) = volfrac;
loop = 0;
change = 1.;
% START ITERATION
while change > 0.01
  loop = loop + 1;
  xold = x;
% FE-ANALYSIS
  [U]=FE(nelx,nely,x,penal);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  [KE] = lk;
  c = 0.;
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = U([nl; n2; n2+1; n1+1],1);
      c = c + (0.001+0.999*x(ely,e1x)^penal)*Ue'*KE*Ue;
      dc(ely,elx) = -0.999 *penal *x(ely,elx)^(penal- 1) *Ue'*KE*Ue;
    end
  end
% FILTERING OF SENSITIVITIES
  [dc]   = check(nelx,nely,rmin,x,dc);
% DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
  [x]    = OC(nelx,nely,x,volfrac,dc);
% PRINT RESULTS
  change = max(max(abs(x-xold)));
  disp([' It.: ' sprintf('%4i',loop) ' Obj.: ' sprintf('%10.4f',c) ...
       ' Vol.: ' sprintf('%6.3f',sum(sum(x))/(nelx*nely)) ...
        ' ch.: ' sprintf('%6.3f',change )])
% PLOT DENSITIES
  colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
end
%%%%%%%%%% OPTIMALITY CRITERIA UPDATE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xnew]=OC(nelx,nely,x,volfrac,dc)
l1 = 0; l2 = 100000; move = 0.2;
while (l2-l1 > 1e-4)
  lmid = 0.5*(l2+l1);
  xnew = max(0.001,max(x-move,min(1.,min(x+move,x.*sqrt(-dc./lmid)))));
  if sum(sum(xnew)) - volfrac*nelx*nely > 0;
    l1 = lmid;
  else
    l2 = lmid;
  end
end
%%%%%%%%%% MESH-INDEPENDENCY FILTER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [dcn]=check(nelx,nely,rmin,x,dc)
dcn=zeros(nely,nelx);
for i = 1:nelx
  for j = 1:nely
    sum=0.0;
    for k = max(i-floor(rmin),1):min(i+floor(rmin),nelx)
      for l = max(j-floor(rmin),1):min(j+floor(rmin),nely)
        fac = rmin-sqrt((i-k)^2+(j-l)^2);
        sum = sum+max(0,fac);
        dcn(j,i) = dcn(j,i) + max(0,fac)*x(l,k)*dc(l,k);
      end
    end
    dcn(j,i) = dcn(j,i)/(x(j,i)*sum);
  end
end
%%%%%%%%%% FE-ANALYSIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [U]=FE(nelx,nely,x,penal)
[KE] = lk;
K = sparse((nelx +1) *(nely+1), (nelx+1) *(nely+1));
F = sparse((nely +l) *(nelx+1),1); U = sparse((nely+1) *(nelx+l),1);
for elx = 1:nelx
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx   +ely;
    edof = [nl; n2; n2+1; nl+l];
    K(edof,edof) = K(edof,edof) + (0.001+0.999*x(ely,elx)^penal)*KE;
  end
end
% DEFINE LOADS AND SUPPORTS (SQUARE PLATE WITH HEAT SINK)
F(:,1)     0.01;
fixeddofs       = [nely/2+1-(nely/20):2:nely/2+1+(nely/20)];
alldofs            [1:(nely+1)*(nelx+1)];
freedofs    = setdiff(alldofs,fixeddofs);
% SOLVING
U(freedofs,:) = K(freedofs,freedofs) \ F(freedofs,:);
U(fixeddofs,:)= 0;
%%%%%%%%%% ELEMENT STIFFNESS MATRIX %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [KE]=lk
KE =     [ 2/3 -1/6 -1/3 -1/6
          -1/6  2/3 -1/6 -1/3
          -1/3 -1/6  2/3 -1/6
          -1/6 -1/3 -1/6  2/3];
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Apêndice A

```
%%%%% Scrip para inicialização da matriz e chamada da função %%%%%

%%%%%% Inicialização da matriz que representa o terreno %%%%%%

ex3(1:50,1:50,1)=2;

%%%%%Obstaculos%%%%%
ex3(1:20,15:17,1)=1;
ex3(40:50,15:17,1)=1;
ex3(1:6,35:37,1)=1;
ex3(10:25,35:37,1)=1;
ex3(1:8,44:46,1)=1;
ex3(12:15,45:47,1)=1;
ex3(30:50,45:47,1)=1;
ex3(25:27,1:15,1)=1;
ex3(35:37,1:15,1)=1;
ex3(25:27,39:45,1)=1;
ex3(15:17,45:50,1)=1;
ex3(20:40,25:27,1)=1;
ex3(35:37,25:35,1)=1;

ex3(5,5,1)=5;%Ponto de partida
ex3(30,40,1)=6;%Ponto de chegada

%%%%% Situacao 2 %%%%%
ex3(:, :, 2) = ex3(:, :, 1);
ex3(30,40,2)=2;
ex3(45,30,2)=6;

%%%%% Situacao 3 %%%%%
ex3(:, :, 3) = ex3(:, :, 2);
ex3(45,30,3)=2;
ex3(5,43,3)=6;

%%%%% Chamada da função %%%%%
[obj vol path caminhos comprimento_caminhos] = top88r2(50,50,5,1,1,ex3);

%%%%% Fim do Script %%%%%



%%%%% Código da função %%%%%
function [obj vol path caminhos comprimento_caminhos] =
top88r2(nelx,nely,nc,rmin,fast,MAP_cenarios)
close all;

caminhos = MAP_cenarios;
comprimento_caminhos = zeros(1, size(MAP_cenarios, 3));

for cont = 1 : size(MAP_cenarios, 3)
```

```
    clearvars -except nelx nely nc rmin fast MAP_cenarios caminhos comprimen-
to_caminhos cont

    tic
    ft=2;
    obsx=[];
    obsy=[];
    sourcepos=[];
    sinkpos=[];

    MAP = MAP_cenarios(:, :, cont);

    mapsize=size(MAP);
    for i=1:mapsize(1)
        for j=1:mapsize(2)
            if MAP(i,j)==1
                obsx=[obsx i];
                obsy=[obsy j];
            elseif MAP(i,j)==5
                sourcepos=[sourcepos j-1 i-1];
            elseif MAP(i,j)==6
                sinkpos=[sinkpos j i];
            end
        end
    end
    %% MATERIAL PROPERTIES
    E0 = 1;
    Emin = 1e-9;
    %% PREPARE FINITE ELEMENT ANALYSIS
   KE = [2/3 -1/6 -1/3 -1/6
        -1/6 2/3 -1/6 -1/3
        -1/3 -1/6 2/3 -1/6
        -1/6 -1/3 -1/6 2/3];
    nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
    edofVec = reshape(nodenrs(1:end-1,1:end-1),nelx*nely,1);
    edofMat = repmat(edofVec,1,4)+repmat([0 nely+[1 2] 1],nelx*nely,1);
    iK = reshape(kron(edofMat,ones(4,1))',16*nelx*nely,1);
    jK = reshape(kron(edofMat,ones(1,4))',16*nelx*nely,1);
    % DEFINE LOADS AND SUPPORTS
    F = sparse((nely+1)*(nelx+1),1);
    % Conduction force
    iF = 0.01;
    F(:,1)=iF;
    for a=1:numel(obsx)
        F((obsy(a)-1)*(nely+1)+obsx(a),1)=0;
    end
    % Robot end points
    qv=sqrt(nelx*nely)/10; %Arbitrary convection component
    for i=1:numel(sinkpos)/2
        nodes = [1 0 -nely -nely-1] + ((nely+1)*sinkpos(2*i-1) +
sinkpos(2*i));
        F(nodes,1)=iF+qv;
    end
    U = zeros((nely+1)*(nelx+1),1);
    fixeddofs=[];
    for i=1:numel(sourcepos)/2
```

```
        fixeddofs = [fixeddofs (nely+1)*sourcepos(2*i-1)+sourcepos(2*i)+2];
        fixeddofs = [fixeddofs nely+1+(nely+1)*sourcepos(2*i-
1)+sourcepos(2*i)+2];
        fixeddofs = [fixeddofs (nely+1)*sourcepos(2*i-1)+sourcepos(2*i)+1];
        fixeddofs = [fixeddofs nely+1+(nely+1)*sourcepos(2*i-
1)+sourcepos(2*i)+1];
    end
    alldofs = [1:(nely+1)*(nelx+1)];
    freedofs = setdiff(alldofs,fixeddofs);
    %% PREPARE FILTER
    iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
    jH = ones(size(iH));
    sH = zeros(size(iH));
    k = 0;
    for i1 = 1:nelx
        for j1 = 1:nely
            e1 = (i1-1)*nely+j1;
            for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
                for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                    e2 = (i2-1)*nely+j2;
                    k = k+1;
                    iH(k) = e1;
                    jH(k) = e2;
                    sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
                end
            end
        end
    end
    H = sparse(iH,jH,sH);
    Hs = sum(H,2);
    %% INITIALIZE ITERATION
    me=1/nelx/nely;
    Mmin = me*max(abs(sinkpos(1)-sourcepos(1)),abs(sinkpos(2)- sourcepos(2)));
    x = repmat(1,nely,nelx);
    xPhys = x;
    loop = 0;
    change = 1;
    obj = []; vol = [];
    cold=0; % Old objectivity function value
    conv_count=0; % Number of consecutive convergences
    %% START ITERATION
    while change > 0.01
        loop = loop + 1;
        %% CALCULATE ALLOWED MASS
        z=1/loop; % z=1/loop^2 for large maps
        Mn=max(z,Mmin);
        %% FE?ANALYSIS
        if (loop <= nc)
            penal = 1;
        else
            penal = min(3,1+0.1*(loop-10));
            sK = reshape(KE(:)*(Emin+0.001+0.999*(xPhys(:))'.^penal*(E0-
Emin)),16*nelx*nely,1);
            K = sparse(iK,jK,sK);
            K = (K+K')/2;
            U(freedofs) = K(freedofs,freedofs)\F(freedofs);
            %% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
```

```
            ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
            c = sum(sum((Emin+0.001+0.999*xPhys.^penal*(E0-Emin)).*ce));
            dc = -0.999*penal*(E0-Emin)*xPhys.^(penal-1).*ce;
            dv = ones(nely,nelx);
            %% FILTERING/MODIFICATION OF SENSITIVITIES
            if ft == 1
                dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
            elseif ft == 2
                dc(:) = H*(dc(:)./Hs);
                dv(:) = H*(dv(:)./Hs);
            end
            %% OPTIMALITY CRITERIA UPDATE OF DESIGN VARIABLES AND PHYSICAL
DENSITIES
            l1 = -0.1; l2 = 1e9; move = 0.2;
            cnt = 0;
            while (l2-l1)/(l1+l2) > 1e-3
                cnt = cnt+1;
                if (cnt>1000)
                    fprintf('Terminate iteration...nn');
                    obj = []; vol = [];
                    return
                end
                lmid = 0.5*(l2+l1);
                xnew = max(0,max(x-move,min(1,min(x+move,x.*sqrt(-
dc./dv/lmid)))));
                for a=1:numel(obsx)
                    xnew(obsx(a),obsy(a))=0;
                end
%%%%% Modificação do valor de mobilidae %%%%%
                 for n=1:mapsize(1)
                    for m=1:mapsize(2)
                        if MAP(n,m)==3
                            xnew(n,m)=0.5*xnew(n,m);
                        end
                    end
                 end
                if ft == 1
                    xPhys = xnew;
                elseif ft == 2
                    xPhys(:) = (H*xnew(:))./Hs;
                end
                if sum(xPhys(:)) > Mn*nelx*nely, l1 = lmid; else l2 = lmid;
end
            end
            change = max(abs(xnew(:)-x(:)));
            x = xnew;
            obj=[obj c];
            vol=[vol mean(xPhys(:))];
            if (abs(c-cold)/abs(c))<0.01, conv_count=conv_count+1;
            else conv_count = 0;
            end
            if (loop>3 && conv_count >=3)
                fprintf('Convergence, cn = %f, cn?1 = %fnn',c,cold)
                change = 0;
            end
            cold = c;
            %% PRINT RESULTS
```

```
            if(~fast || change <= 0.01)
                fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3fnn\n'...
                    ,loop,c,mean(xPhys(:)),change);
                %% PLOT DENSITIES
                if change<= 0.01,
                    xPhys=round(xPhys+0.11*MAP);
%%%%% Trajetória obtida para cenário de número igual a cont e cálculo do com-
primento%%%%%
                    caminhos(:, :, cont) = xPhys;
                    comprimento_caminhos(cont) = sum(sum(xPhys == 1));
                    path = xPhys;
                end
                figure;
                imagesc(xPhys+0.25*MAP);
                caxis([0 1]); axis equal; axis off; drawnow;
            end
        end
        t=toc;
        fprintf('Time taken: %4.3f Average/iteration: %4.4fnn',t,t/loop);
        path = xPhys;
    end
end
%%%%% Comparação dos comprimentos dos caminhos obtidos %%%%%
    indice_menor_caminho = find(comprimento_caminhos ==
min(comprimento_caminhos));
for i = 1 : length(indice_menor_caminho)
    MAP = MAP_cenarios(:, :, indice_menor_caminho(i));
    figure;
    imagesc(caminhos(:, :, indice_menor_caminho(i)) + 0.25*MAP);
    caxis([0 1]); axis equal; axis off; drawnow;
    title(['Menor Caminho: Rota ' num2str(indice_menor_caminho(i)) ', '
num2str(comprimento_caminhos(i)) ' unidades de espaço']);
end
end
```