# Uma aplicação de algoritmos genéticos multiobjetivos NSGA-II na geração de rotas de fuga em ambientes fechados.

Sérgio Ricardo Argolo Queiroz Filho
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N
Barbalho, Salvador, Bahia
+55 (71) 99306-3597
srargolo@gmail.com

Frederico Barboza
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N
Barbalho, Salvador, Bahia
fred.barboza@gmail.com

## Resumo

Este trabalho apresenta uma aplicação de algoritmo evolutivo multiobjetivo NSGAII gerando rotas de fugas de pessoas em ambientes fechados. O algoritmo considera informações da quantidade e localização das pessoas e dos pontos de saída. Estas rotas são personalizadas para cada pessoa contida neste local. Os objetivos são: minimizar o tempo de evacuação e realizar o balanceamento do número de pessoas nas saídas. Conjecturamos que o algoritmo, devidamente parametrizado, gera soluções que convergem para aquelas consideradas próximas das Pareto-ótimas ou eficientes. Foram realizados testes estatísticos para validação dos resultados obtidos.

## Palavras-Chave

Ambientes fechados, rotas de fuga, otimização multiobjetiva, algoritmo genético, NSGA-II.

## 1. INTRODUÇÃO

A natural evolução humana e busca por melhoria de vida, forçaram o ser humano a conviver a maior parte do tempo de suas vidas em locais fechados, em grandes centros urbanos, como edifícios, hospitais e fábricas. Para Narciso, Maslin-kiewicz e Freitas [3], muitas pessoas chegam a passar mais de 80% de seu tempo em ambientes fechados, enquanto Castro [2] reporta que esse número poderá chegar a 92% em grandes centros urbanos. Muitos desses locais são bem projetados e construídos de forma a garantir o bem-estar e o desenvolvimento dos indivíduos neles contidos. Porém, em muitos casos, o fator segurança humana em situações atípicas é negligenciado, por diversos motivos como: custo da obra, falta de planejamento, menosprezo por acidentes e incidentes pós-obra. Nota-se que, durante a construção desses locais, há uma grande preocupação com incidentes e acidentes, mas na maioria das vezes, no projeto da edificação não conseguimos observar muita importância dada à segurança das pessoas que habitarão e/ou utilizarão estes espaços diariamente.

Esta realidade evidencia a necessidade de termos maiores preocupações referentes à segurança das pessoas em ambientes fechados. Espaços para eventos (shows de bandas, casamentos, aniversários, por exemplo), são casos típicos que trazem reflexões quanto à necessidade da criação de soluções que possam salvar vidas e até evitar acidentes, como consequências de emergências quando pessoas podem ser pisoteadas, devido a todos irem para a mesma saída ou dirigirem-se a locais sem saídas, onde o risco de perderem a vida é iminente. Emergência é a "situação que exige uma intervenção imediata de profissionais treinados com equipamentos adequados, para que danos e prejuízos sejam evitados ou minimizados" [4]. Lugon [4] explica que o chefe da equipe de resgate é o responsável por definir as técnicas mais adequadas de acordo com as características de uma emergência. Para Lugon [4], a estratégia é a decisão sobre a escolha e forma de emprego do conjunto de técnicas, as quais serão utilizadas no combate à emergência, seja ela de combate a incêndio, busca e salvamento, atendimento pré-hospitalar ou todas elas combinadas. Mas, nada adianta se, como ocorre com frequência, esta ajuda especializada em salvamentos, tiver contratempos que retardem seu tempo de ação, mesmo com as melhores definições estratégicas de salvamento de pessoas. Diante disto, necessitamos de técnicas que tratem do gerenciamento de situações de emergências em áreas fechadas, em paralelo aos atuais métodos de salvamento a posteriori, mas que sejam planejados antes destas situações ocorrerem.

A computação possui grande utilidade em diversas áreas das atividades humanas. Assim, podemos pressupor que vidas poderiam ser salvas em emergências ocorridas em ambientes fechados se houvesse o gerenciamento emergencial com o auxílio de tecnologias computacionais. Esta ferramenta computacional poderia fornecer informações de direcionamento para melhores rotas de fuga de forma ordenada, em particular para o número de pessoas que se dirigem para cada saída e o tempo de resposta para conseguir esvaziar o local. Em uma hipótese superior, as próprias pessoas poderiam resguardar suas vidas, e até das demais existentes no local, se as indicações fossem personalizadas e atualizadas constantemente sobre tais rotas de fuga, específicas para cada indivíduo em determinada localização do ambiente em questão, nas situações emergenciais.

Informações sobre melhores rotas de fuga disponíveis às pessoas que se encontram em ambientes fechados, podem ser mais eficazes do que os resgates especializados, evitando, assim, catástrofes ou perdas de vidas humanas. Informações precisas e atualizadas constantementes, sobre pessoas imersas em ambientes *indoor*, podem ser fundamentais para a manutenção da vida humana.

Observamos que na maioria das edificações não há um plano eficiente direcionado a cada ambiente fechado, nem às pessoas nestes contidos. Este plano de contingência poderia indicar melhores rotas de fuga, evitando, assim, grande pânico e desordem nos momentos de evacuações emergenciais. A computação pode ajudar através do uso de ferramentas que calculem as melhores rotas de fuga, fazendo o balanceamento e minimizando a concentração excessiva de pessoas nas saídas de emergência e também minimizando o tempo de fuga. Este é um problema de otimização multiobjetivo, pois existem mais de um objetivo a ser alcançado em um único cenário. Este problema possui uma complexidade computacional do tipo NP, ou seja, existe a necessidade de obter a melhor solução através da validação de cada uma das soluções (verificar qual o tempo de fuga para cada rota existente, para cada saída de cada indivíduo inserido no problema). A depender destas variáveis, quantidade de pessoas, saídas e rotas de fuga, é computacionalmente impossível de se resolver este problema em um tempo polinomial. Em muitos casos, o tempo despendido é superior ao tempo disponível para a tomada de decisão. Através da utilização de heurísticas de aproximação, como um algoritmo genético, existe a possibilidade de se obter resultados ótimos aproximados em tempo hábil.

Neste trabalho descrevemos uma implementação do algoritmo genético multiobjetivo NSGA-II a fim de minimizar duas grandezas: balanceamento das saídas e tempo de evacuação. O objetivo é gerar recomendações de rotas de fuga específicas para cada pessoa contida no ambiente fechado. Foram consideradas a quantidade de pessoas contidas em cada subambiente e as distâncias até as saídas. Foram geradas soluções consideradas ótimas. Estas soluções foram utilizadas em testes estatísticos. Conjecturamos que o uso de algoritmos genéticos multiobjetivos NSGA-II, podem realizar os cálculos adequados, em tempo útil, para obtenção de tais resultados. Os dados resultantes podem ser utilizados em uma ferramenta de gestão de evacuação de pessoas em lugares fechados.

Além desta introdução, este trabalho está organizado como segue: a Seção 2 discute os principais tópicos relacionados à pesquisa desenvolvida neste trabalho, um referencial teórico. A seção 3 trata sobre trabalhos relacionados e as features deste projeto. A Seção 4 descreve o trabalho proposto. A Seção 5 demonstra a metodologia do trabalho em questão. Seção 6 discorre sobre as lições obtidas neste projeto. Já a seção 7, trata sobre a bibliografia.

## 2. REFERENCIAL TEÓRICO

Esta seção discute os principais tópicos relacionados à pesquisa desenvolvida neste trabalho, à otimização multiobjetiva, ao algoritmo NSGA-II e alguns termos utilizados.

### 2.1 Otimização Multiobjetivo

A otimização pode também ser definida como a necessidade de eficiência, que pode ser dividida nas etapas de reconhecimento das alternativas e a decisão [7]. Primeiramente, é reconhecida numa tarefa ou problema a possibilidade de optar entre várias hipóteses e, em seguida, é decidida por aquela que se considera a melhor opção [8].

A decisão pode ser qualitativa quando a escolha é feita através do bom senso, ou quantitativa quando a decisão é baseada em métodos matemáticos. No entanto, no processo de tomada de decisões é de grande auxílio o uso de alguns métodos ou técnicas para realizar tarefas ou resolver problemas.

No mundo real, a grande maioria dos problemas apresenta mais de um objetivo a serem otimizados que são, na maioria das vezes, conflitantes entre si, ou seja, a melhoria de um ou mais objetivos causa, consequentemente, a deterioração de outro(s)[7].

Em Deb [1] é apresentado um exemplo bastante básico que possui objetivos conflitantes.

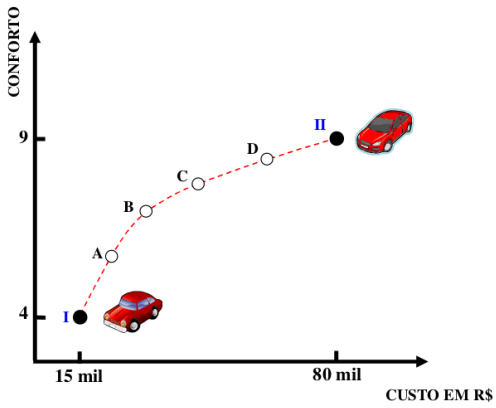


**Figura 1: Exemplo de gráfico com caraterísticas de automóveis.**

Considerando a ilustração, um automóvel (solução I) e outro (solução II) com os respectivos valores de mercado: R$ 15 mil e R$ 80 mil. Nesta situação, supondo que o único objetivo é minimizar o custo do automóvel, a solução ótima escolhida seria a representada por I. Na vida real, é evidente que esta situação não admite um único objetivo, ou seja, deve-se considerar também o grau de conforto do automóvel, dentre outros diversos fatores. Neste caso, um automóvel com menor custo, também será menos confortável. A figura 1 indica que o carro mais barato possui um nível de conforto igual a 4, contrária à solução II que possui um grau de conforto igual a 9. Pode-se verificar que entre as soluções extremas (I e II) existem outras soluções que proporcionam uma variação de custo e conforto (linha tracejada). Pode-se notar que entre todas as soluções presentes, não é possível dizer que uma solução é melhor que outra, ou seja, nenhuma solução que tenha menor custo e conforto pode ser considerada superior à outra com maior custo e conforto. A escolha do melhor automóvel representado por estas soluções só pode ser feita levando em consideração os objetivos do comprador. Portanto, isso faz com que esses problemas apresentem um conjunto de soluções ótimas.

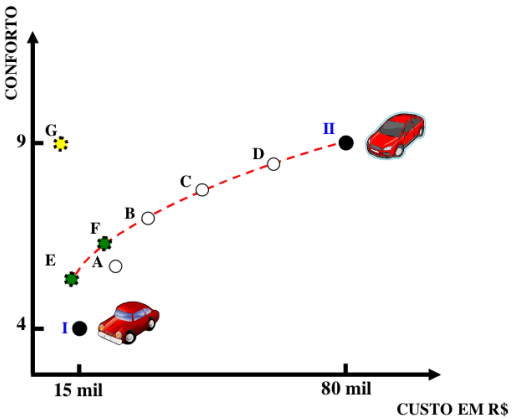


**Figura 2: Gráfico com características de outros automóveis.**

Considerando a mesma situação e acrescentando mais 3 tipos de carros (soluções E, F e G), embora comprar um carro com baixo custo e alto conforto seja uma condição ideal, esta condição não é real para o mercado. Portanto, a solução G com baixo custo e elevado conforto é uma opção dita absurda e deve ser desconsiderada. Existem soluções que são superiores às demais, isto é, apresentam maior conforto e custo menor ou igual, como ocorre com as soluções E e F que superam, respectivamente, as soluções I e A. Estas soluções são chamadas de não-dominadas, enquanto que, as soluções que são superadas por, pelo menos, uma outra solução, são denominadas dominadas. Assim, no exemplo da Figura 2, quem deseja comprar um automóvel, diante de todas as opções válidas apresentadas, a melhor opção é escolher entre as soluções E, F, B, C, D e II. Assim, a técnica que encontra o conjunto das soluções não-dominadas que, entre essas, possa sugerir aquela que melhor atenda as necessidades, é chamada de otimização multiobjetivo.

## 2.1.1 Formas de tratamento da Otimização Multiobjetivo

Duas ideias foram esquematizadas em Deb [1], considerando que o problema possui duas funções objetivo. A primeira sugere que, definidas as prioridades e pesos entre os vários objetivos, busca-se a solução ótima. Já a segunda explicita que, não havendo informação adicional, encontra-se o conjunto de soluções não-dominadas, depois escolhe-se uma solução deste conjunto.

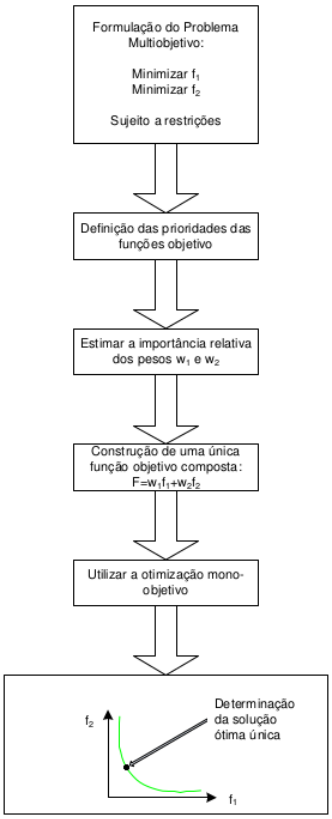


**Figura 3 : Esquematização encontrada em Deb[1].**

Geralmente é bastante difícil estabelecer o nível relativo da importância dos objetivos. Assim, a difícil questão desta primeira abordagem é como definir todas as prioridades e pesos dos problemas que, fatidicamente, se conhecem pouco. As diversas funções objetivo do problema são transformadas em uma única, bastando utilizar uma técnica de otimização mono objetivo.

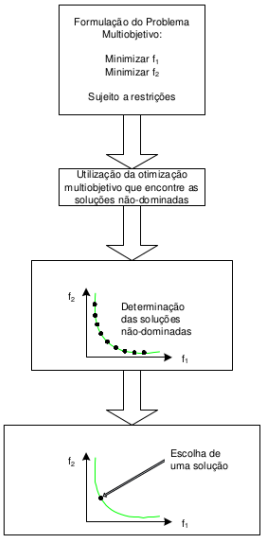


**Figura 4 : A outra esquematização encontrada em Deb[1].**

Existem algoritmos evolucionários que trabalham com os problemas baseando-se nesta abordagem da Figura 3, possibilitando a obtenção de um grande número de soluções não-dominadas para escolha entre estas a melhor opção. Porém, em problemas multiobjetivos do mundo real existe a dificuldade na obtenção de informações quantitativas e qualitativas inerentes ao problema. Conclui-se que a segunda abordagem, apresentada na Figura 4, possui maior valor e faz com que os Algoritmos Evolucionários sejam preferidos em relação aos métodos clássicos[5].

## 2.1.2 Descrição do problema de Otimização Multiobjetivo

A formulação genérica de um problema de otimização multiobjetiva, envolve a minimização ou maximização de algumas funções objetivos que estão sujeitas a uma determinada quantidade de restrições[2].

$$\text{Otimizar } Z = f_k(\vec{x}),\ \text{para } k = 1,2,\ldots,l \text{ com } l \geq 2$$

$$\text{Sujeito a}$$

$$g_j(\vec{x}) \leq 0,\ \text{para } j = 1,\ldots,q$$

$$h_j(\vec{x}) = 0,\ \text{para } j = q+1,\ldots,m$$

$$l(i) \leq x_i \leq u(i), \quad 1 \leq i \leq n$$

**Equação 1: pseudocódigo da otimização multiobjetivo**

Nesta expressões, o que se busca é encontrar um ponto $\vec{x} = (x_1, \ldots, x_n)$ que satisfaça o conjunto de restrições e otimize as funções

objetivo envolvidas no problema. Assim, com a presença de mais de uma função objetivo, surge um novo espaço importante denominado espaço objetivo. Logo, é feito o mapeamento do espaço das variáveis de decisão com o seu correspondente espaço objetivo.

A maneira mais interessante para obter as soluções ótimas, é encontrar um conjunto de soluções que sejam não-dominadas. Quando se refere às soluções não-dominadas, essas superam outras soluções levando em conta os valores das funções objetivo. Sendo assim, o comportamento de uma solução no espaço objetivo é o que determina se esta é não-dominada ou dominada. Entende-se que a relação entre as soluções mapeadas no espaço objetivo irá guiar a escolha de pontos desejáveis no espaço das variáveis. Essa relação é traduzida pela comparação entre cada uma duas soluções mapeadas, levando-se em consideração cada função objetivo do problema [7].

### 2.1.3 Dominância

Visando uma comparação mais justa entre os vários objetivos, foi criado o conceito de dominância [5] que é uma característica dos algoritmos multiobjetivos. Essa classe de algoritmos que leva em consideração a otimização de mais de um objetivo, sendo estes normalmente conflitantes. O critério de dominância segue a seguinte regra :

Dados dois indivíduos p e q pertencentes a uma mesma população P, então:

- Um indivíduo *p* domina um indivíduo *q*, se no mínimo o valor em um dos objetivos de *p* é melhor que o mesmo objetivo em *q* e o restante dos valores dos objetivos de *p* não pode ser pior que o restante dos mesmos valores nos objetivos em *q*. Isso significa dizer que *p* não pode possuir nenhum objetivo com menos qualidade do que *q*.

Ao final de cada análise, um determinado grupo de indivíduos é classificado como pertencente a uma categoria específica denominada *front. A*o ser concluído o processo classificatório, todos os indivíduos estarão inseridos em um dos *n fronts*. O *front 1* é constituído de todas as soluções não-dominadas. Já o *front 2* pode ser conseguido considerando todas as soluções não-dominadas, excluídas as soluções do front 1. Para determinação do front 3, excluem-se as soluções previamente classificadas no front 1 e 2, e assim por diante até que todos os indivíduos tenham sido classificados em algum front.

### 2.1.4 Soluções Pareto-Ótimas

Dado um conjunto de soluções P, o conjunto de soluções não-dominadas P' é aquele que contém elementos não-dominados por qualquer outro elemento do conjunto P. Então, quaisquer soluções de P', são não-dominadas entre si e qualquer solução das demais do conjunto P são dominadas por pelo menos um elemento de P'. Caso o conjunto P seja o próprio espaço de busca, então o conjunto P' é chamado de conjunto Pareto-ótimo [8].

Há diferença entre um conjunto de soluções não-dominadas e um conjunto Pareto-ótimo. Um conjunto de soluções não-dominadas é definido no contexto de uma amostra do espaço de busca, enquanto o conjunto Pareto-ótimo é definido em relação a todo espaço de busca.

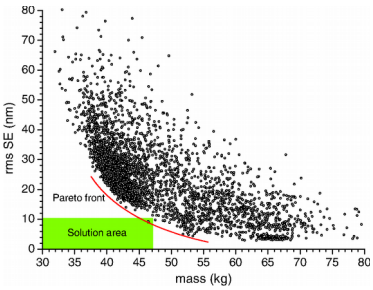


**Figura 5: Exemplo da Frente de Pareto.**

## 2.2 Algoritmos genéticos

Algoritmos genéticos foram introduzidos por J.H. Holland [6] e são baseados na teoria do mecanismo de seleção natural proposto por Charles Darwin. Estes permitem que, a partir de um conjunto de soluções iniciais, com o passar das gerações (número de iterações do algoritmo genético), este conjunto possa atingir resultados ainda melhores do que os das gerações anteriores. Um algoritmo genético convencional é constituído por quatro etapas:

- Geração de população inicial
- Cruzamento
- Mutação
- Seleção.

Graças aos operadores de cruzamento, mutação e seleção, é possível melhorar os resultados entre as gerações.

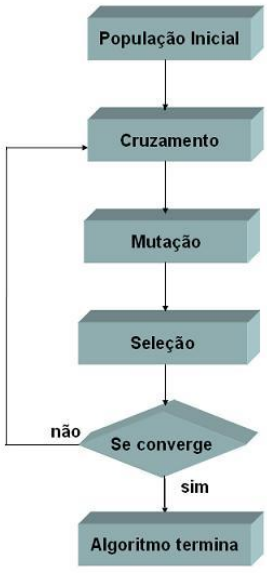


**Figura 6: Fluxograma de um algoritmo genético clássico**

### 2.2.1 Geração da população inicial

A população de um algoritmo genético é o conjunto de indivíduos que estão sendo cogitados como solução e que serão usados para criar o novo conjunto de indivíduos. O tamanho da população

pode afetar o desempenho geral e a eficiência dos algoritmos genéticos. Populações muito pequenas têm grandes chances de perder a diversidade necessária para convergir para uma boa solução, pois fornecem uma pequena cobertura do espaço de busca do problema. Entretanto, se a população tiver um número exagerado de indivíduos, o algoritmo poderá perder grande parte de sua eficiência pela demora em avaliar a função de aptidão de todo o conjunto a cada iteração, além de exigir mais recursos computacionais.

O uso de algoritmo genético como ferramenta para solução de problemas, inicia-se na representação desses problemas de maneira que os algoritmos genéticos possam trabalhar adequadamente sobre eles. Uma das principais formas é representar cada atributo como uma sequência de bits e o indivíduo, como a concatenação das sequências de bits de todos os seus atributos. Mas existem outras formas. A codificação usando o próprio alfabeto do atributo que se quer representar (letras, códigos, números reais, etc.) para representar um indivíduo também é muito utilizada. Diversas outras formas são possíveis. Normalmente a forma mais apropriada está fortemente ligada ao tipo de problema.

Um outro ponto importante é a avaliação de cada indivíduo, realizada através de uma determinada função - o valor de aptidão de cada indivíduo da população. Esta etapa é a mais importante de qualquer algoritmo genético. É por meio desta função que se mede quão próximo um indivíduo está da solução desejada ou quão ótima é esta solução. Necessita-se de que esta seja muito representativa e diferencie, na proporção correta, as más soluções das ótimas. Caso haja pouca precisão na avaliação, uma ótima solução pode ser excluída durante a execução do algoritmo, além de demandar mais tempo explorando soluções possivelmente pouco promissoras.

Após atribuir a cada indivíduo um valor de aptidão, é necessário aplicar um método de seleção, e existem vários métodos para selecioná-los, sobre os quais serão aplicados os operadores genéticos. Entre as diversas formas de seleção, há o método de seleção por Roleta e por Torneio. No método de seleção por Roleta, cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão. Assim, para indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos indivíduos de aptidão mais baixa, é atribuída uma porção relativamente menor. Mas um dos problemas encontrados pode ser o tempo de processamento, pois este exige duas passagens por todos os indivíduos da população. Já o método de seleção por Torneio possui a dinâmica em que um número *n* de indivíduos da população é escolhido aleatoriamente para formar uma sub-população temporária. Deste grupo, o indivíduo selecionado dependerá de uma probabilidade *k* definida previamente. Sendo assim, este método é mais utilizado em relação ao anterior, pois oferece a vantagem de não exigir que a comparação seja feita entre todos os indivíduos da população [9].

### 2.2.2 Cruzamento

Os indivíduos existentes na população, respeitando uma probabilidade inicialmente imposta, irão trocar sequências, aleatoriamente escolhidas, de suas informações gerando novos indivíduos que herdam características dos indivíduos anteriores. A depender da situação, pode ser bastante semelhante à união cromossômica para formar novas combinações de genes humanos. Indivíduos detectados como repetidos, podem ser descartados, mesmo após o cruzamento ajudando a manter a diversidade das soluções.

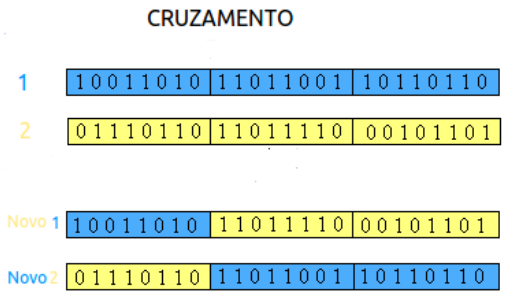


**Figura 7: Exemplo de cruzamento**

Podemos observar na Figura 7 um tipo de cruzamento, onde partes dos cromossomos são trocados entre sí gerando novos indivíduos. O primeiro alelo do indivíduo 1 foi trocado com o primeiro alelo do indivíduo 2, gerando dois novos indivíduos: *Novo1* e *Novo2*.

### 2.2.3 Mutação

Nesta operação, novos indivíduos não são gerados à população, ou seja, não há um aumento numérico à população, como ocorre por intermédio do operador de cruzamento, mas ele apenas modifica esses indivíduos que já existem transformando-os em outros indivíduos diferentes. Cada indivíduo, obedecendo a uma probabilidade inicialmente imposta, irá em alguns pontos de sua sequência, mudar de valor. Indivíduos repetidos podem ser descartados da população, ou seja, caso o indivíduo resultante da mutação já exista, este será descartado e não será considerado válido, mantendo, assim, a diversidade das soluções. Conforme o exemplo da Figura 8, o sexto alelo tem seu valor mudado. Isso fará com que o algoritmo consiga uma maior diversidade genética que a função de cruzamento, sozinha, não seria capaz de fornecer. O operador de mutação permite uma maior variabilidade genética e uma maior exploração do universo de possibilidades de configurações. Também evita que o algoritmo fique estacionado em mínimos locais, fornecendo maior vantagem para o operador de mutação, pois ele permite uma variação brusca das características que o indivíduo possui através da alteração de partes aleatoriamente selecionadas dos seus genes.

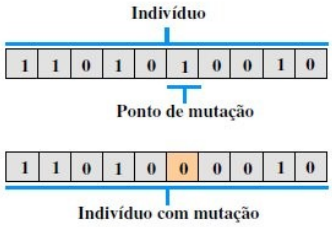


**Figura 8: Exemplo de mutação.**

### 2.2.4 Seleção

Nessa fase, que será utilizada uma função objetivo para dar ao algoritmo padrões de seleção dos melhores indivíduos, logo, é a etapa mais relevante do algoritmo genético.
A função objetivo é essencial para propagar às gerações futuras os melhores resultados de cada geração. Neste caso, somente as melhores soluções continuarão a existir na próxima geração.
Existem vários métodos para selecionar os indivíduos sobre os quais serão aplicados os operadores genéticos. Alguns destes métodos:

- Seleção por ranking: os indivíduos da população são ordenados de acordo com seu valor de adequação e, então, sua probabilidade de escolha é atribuída conforme a posição que ocupam.
- Seleção por giro de roleta: cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão. Assim, para indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos indivíduos de aptidão mais baixa, é dada uma porção relativamente menor.
- Seleção por torneio: A ideia desta etapa é promover um torneio entre um grupo de $n$ ( $n \geq 2$ ) indivíduos aleatoriamente tomados na população. Assim, o indivíduo que vencer este torneio (indivíduo com melhor valor de aptidão entre o grupo), é selecionado para fazer parte da geração da nova população, enquanto os demais indivíduos do grupo são descartados. O processo de seleção termina quando se realiza uma quantidade de torneios igual ao tamanho da população [9].

### 2.2.5 Convergência

A convergência é fortemente influenciada pela construção do algoritmo genético: representação, decodificação, avaliação, operadores, técnicas, parâmetros e implementação podem facilitar a busca de soluções ótimas ou sub-ótimas. Existem alguns métodos a fim de alcançar soluções em espaços complexos. Um deles, e o mais simples, utiliza a quantidade de gerações como condição de parada. Assim, quando o algoritmo alcança um número $n$ de gerações, o mesmo finaliza sua execução. Já o segundo método de convergência, verifica alterações significativas dos indivíduos da população entre $n$ gerações. Se entre as gerações não for verificada uma variação significativa da qualidade, então, existe a indicação de convergência, e o algoritmo é finalizado. Caso contrário, (se a taxa de variação calculada pelo algoritmo supera a taxa de variação percentual inicialmente pré-definida), o algoritmo segue sua execução normal em direção à nova geração. O método de convergência é verificado durante todas as próximas gerações até que o algoritmo demonstre a convergência sobre uma uma determinada solução ótima.

## 2.3 NSGAII

O algoritmo NSGA II (Non-dominated Sorting Genetic Algorithm II) foi escolhido pois é um algoritmo multiobjetivo comercialmente aceito e utilizado em larga escala. Além disso, é baseado em Algoritmos Genéticos [1] e implementa o conceito de dominância. Classifica a população total em fronts de acordo com o grau de dominância. Os indivíduos que estão localizados no primeiro front são considerados as melhores soluções daquela geração. Os do último front encontram-se nas piores soluções. Assim, podem-se encontrar resultados mais consistentes (pontos próximos da região de Pareto) e que se adaptam melhor ao tipo do problema, mais rapidamente (menor número de gerações). Dois aspectos são importantes na solução de problemas multiobjetivos:

- Dividir a população em diferentes níveis (fronts), utilizando o critério de dominância;
- Indivíduos do front n são melhores do que indivíduos do front n+1.

Com a dominância, o algoritmo agrega o conceito de elitismo. O elitismo classifica a população total em diferentes categorias de qualidade, ao invés de tratá-las como pertencentes a um único grupo. Assim, é possível que o algoritmo priorize aqueles que foram melhores classificados.
O funcionamento do NSGA II se destaca por possuir dois mecanismos importantes no processo de seleção:

- *Fast Non-Dominated Sorting*
- *Crowding Distance*

Inicialmente, o que existe é uma população inteira ainda não classificada. Esta passará por um processo em que será atribuído a cada indivíduo um grau de dominância em relação a todos os outros indivíduos da população inteira. Após, ocorre a comparação de uns com os outros, e assim classificando-os de acordo com o critério de dominância descrito inicialmente. Logo após os valores de dominância terem sido atribuídos a todos da população, esses indivíduos serão classificados em *fronts (*de acordo com os seus valores de dominância). Os melhores indivíduos são classificados no primeiro *front.* Os piores serão classificados como pertencentes ao último *front*. Assim sucessivamente até que não haja mais indivíduos a serem classificados. Ao término da classificação de todos os indivíduos dentro de um *front*, eles serão classificados pelo *Crowding Distance*. Esse operador irá ordenar cada indivíduo de acordo com a sua distância em relação aos pontos vizinhos no mesmo *front* (em relação a cada objetivo). Quanto mais distantes desse ponto central, maior a probabilidade de serem selecionados. Esse operador permite que haja um melhor espalhamento dos resultados, evitando-se assim aglomerações de soluções sobre um mesmo ponto e uma convergência prematura.

### 2.3.1 População

O tamanho da população afeta o desempenho global e a eficiência do algoritmo. Com uma população reduzida, o desempenho pode ser inferior, pois deste modo a população fornece pouca cobertura do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura representativa do domínio do problema e previne convergências prematuras para soluções locais ao invés de globais. Porém, para utilizar grandes populações, são necessários maiores recursos computacionais, ou que o algoritmo trabalhe por um período de tempo muito maior, o que não desejamos que aconteça.

Diferentemente de outras etapas de outros algoritmos genéticos, a geração da população inicial é realizada somente no início do algoritmo. Esta acontece uma única vez e não se repete ao longo das gerações seguintes. Indivíduos que codificam um grupo de $n$ visualizações distintas são aleatoriamente gerados dentro do algoritmo e armazenados em uma estrutura de dados.
Após a geração de cada indivíduo, pode ocorrer a verificação se não há repetições. Este procedimento serve para evitar a repetição de indivíduos, pois é fundamental para o ideal funcionamento do algoritmo que haja diversidade de soluções. Assim, permite-se uma maior variabilidade genética durante as gerações. Também pode contribuir para que soluções melhores sejam encontradas. Se um indivíduo atualmente gerado já existe na população, este poderá ser descartado e um novo indivíduo será criado, buscando completar o número de indivíduos da população inicial.

### 2.3.2 Fast Non-Dominated Sorting

O *Fast Non-Dominated Sorting* é um operador de seleção executado em duas etapas.
A primeira etapa irá analisar todos os indivíduos da população total *P.* Depois, serão comparados para classificá-los de acordo

com o grau de dominância *np* (número de indivíduos que dominam *p*, em que *p* é um individuo da população *P*). Dessa forma, se um indivíduo *p é* dominado por um número *X* de indivíduos da população total *P*, o seu valor correspondente de *np é* igual a X. Se ao final desta etapa o indivíduo possuir o valor de *np* igual a 0, significa que esse indivíduo não é dominado por ninguém da população total *P*. Estes indivíduos farão parte do primeiro *front*, no qual estão os melhores indivíduos de toda a população atual.
A segunda etapa separará cada indivíduo em diferentes categorias (os *fronts*). Esta se dará de acordo com os seus valores de dominância, indicados pelos seus respectivos valores de *np*. Cada indivíduo incluído em um dos fronts são retirados do contexto do sistema, decrementando os valores de *np* de cada indivíduo dominado por esses. Isso repetir-se-á até que não sobrem mais indivíduos na população restante.

```
1.  para cada p (pertencente a) P
2.      para cada q (pertencente a) P
3.          se (p domina q) então
4.              Armazena q em Sp (Sp <- q)
5.          senão se (q domina p) então
6.              Incrementa np (np <- np+1)
7.  se np (é igual a) 0 então
8.      Armazena p em F1 (F1 <- p)
```

**Algoritmo 1: Pseudo-código da etapa um do fast non-dominated sorting.**

O funcionamento do pseudo código descrito acima (Algoritmo 1), ocorre da seguinte maneira:

1. Um indivíduo *p* é selecionado da população *P* (linha 1)
1. Para cada indivíduo *q*, restante na população *P* (linha 2), é verificado se o individuo *p* domina o individuo *q* (linha 3)
2. Se confirmado, o indivíduo *q* será armazenado em *Sp* (na lista dos indivíduos dominados por *p*) (linha 4)
3. Negativamente, se *q* dominar *p* (linha 5) o valor de *np* é incrementado (pois *np* é um contador de indivíduos dominantes de *p*) (linha 6)
4. É testado se o *np é* igual a 0 (linha 7), sendo verdade, significará que *p* não foi dominado por ninguém da população. Este fará parte do primeiro *front* (linha 8).

O *fast-nondominated-sorting* comparará indivíduo a indivíduo para determinar o grau de dominância *np* de cada componente da população; quanto menores os valores de *np* encontrados, melhores soluções irão representar.

```
1.  Inicializa uma variável i com o valor 1 (i<-1)
2.  enquanto Fi (for diferente de) 0
3.      H = 0
4.      para cada p (pertencente a) Fi
5.          para cada q (pertencente a) Sp
6.              decrementa o valor de nq (nq <- nq-1)
7.              se nq ( é igual a ) 0 então H <- q (insere em H o valor de q)
8.                  Incrementa o valor de i em uma unidade (i <- i+1)
9.              Armazena os valores de H no Fi (que será o front atual no próximo loop)
```

**Algoritmo 2: Pseudo-código da segunda etapa do fast non-dominated sorting.**

Observa-se uma variável *I* (contador do número de *fronts*) que é inicializada com o valor unitário (linha 1). Enquanto houver novos *fronts* sendo gerados (se o tamanho de *Fi* - o atual *front* - for diferente de 0), a execução continuará (linha 2).

A variável *H* é inicializada (variável temporária auxiliar dos próximos *fronts*) com o valor 0 (linha 3). Nas próximas linhas, 4 e 5, um indivíduo *p* do *front Fi* é selecionado, e uma verificação é realizada para encontrar quais os indivíduos em *q*, dominados por *p* (encontrados na lista *Sp*) e os valores dos *nq*, desses indivíduos, são decrementados, com o objetivo de retirar do contexto os indivíduos que foram classificados anteriormente no *front* antecessor. O decréscimo dos valores de *nq se o*bjetiva da aproximação do valor 0 - significa dizer que em uma próxima iteração, os indivíduos com *nq* igual a 0 farão parte do novo *front* (linhas 6 e 7). Na linha 8 o valor de *I* será incrementado e os valores armazenados em *H* serão copiados para o novo *Fi* - o novo *front* - (linha 9).

### 2.3.3 Crownding distance

Este é o operador de diversidade usado no NSGA II. O objetivo é garantir um maior espalhamento dos resultados ao longo da linha de Pareto. Pretende-se, assim, evitar a concentração de soluções em um mesmo ponto ou região. Também é utilizado como método de ordenação dos indivíduos de um mesmo *front*.
O Crowding Distance utiliza como métrica a distância de cada indivíduo aos indivíduos mais próximos. Este algoritmo calcula a distância média entre um ponto central *i*, selecionado dentro da população e dois pontos localizados nas extremidades do ponto central *(i-1)* e *(i+1)*. A partir de um ponto central, este operador de diversidade, encontrará pontos extremos e priorizará outros mais distantes durante o processo de seleção. Busca-se assim, espalhar os resultados ao longo da frente de Pareto. A disposição dos pontos extremos formam um cuboide em relação ao ponto central.

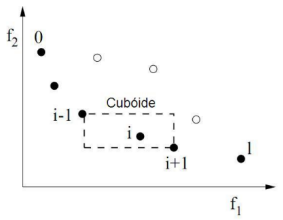


**Figura 9: Representação do cálculo da distância – Operador de diversidade.**

```
1.  distância de cada ponto no conjunto S :
2.  atribuição do crowding-distance(S)
3.  l=|S| (|S| representa o número de elementos do conjunto S e que será atribuído à l)
4.  para cada i, do conjunto S[i]distancia = 0
5.  para cada objetivo m :
6.      ordenar(S, m) (ordena cada indivíduo do grupo S em relação a cada objetivo m)
7.  S[0]distancia e S[l]distancia <- valor alto
8.  para i = 1 até (l-1) (para todos os outros pontos)
9.      S[i]distancia = S[i]distancia + (S[i+1].m - S[i-1].m)
```

**Algoritmo 3: Pseudocódigo do algoritmo "Crowding Distance".**

O funcionamento do pseudo código descrito acima (Algoritmo 3), ocorre da seguinte maneira:

1. Na linha 3, é criada uma variável *l*, que será um contador de valores existentes no conjunto de entrada *S*.
2. São inicializados todos os valores do conjunto *S[i]distancia* (vetor que conterá todas as distâncias de cada indivíduo do conjunto *S*) com o valor 0 (linha 4).

3. Este operador levará em consideração os *m* objetivos em questão, ordenando em relação à cada um deles (linha 6).
4. São inicializadas as posições *S[0]distancia* e a ultima posição *S[l]distancia* do vetor distância, com um valor muito alto (linha 7).
5. Na linha 8 em diante, existe um laço que chamará o cálculo do valor médio dos pontos em relação ao ponto central *i*.
6. Segundo a fórmula da linha 9, *S[i+1].m* e *S[i-1].m*, significam a posição *i+1* e *i-1* do vetor *S* em relação ao objetivo *m*.
7. Os resultados serão ordenados dando preferência às soluções com distâncias médias maiores.

O algoritmo compõe-se de uma estratégia para que esses indivíduos, que se situam nas extremidades, sejam sempre selecionados (o crowding distance prioriza aqueles que possuem os maiores valores de distância). Este visa encontrar soluções diversificadas e que não se encontrem concentradas em uma região limitada do gráfico, que permita uma ampla visão sobre as melhores soluções na tomada de decisões.

## 3. Trabalhos relacionados

Considerando que este tema é bastante específico, não conseguimos encontrar nenhum trabalho nesta área de gerenciamento emergencial *indoor* com abordagem multiobjetiva. Mas, algumas iniciativas possuem características relevantes para nosso entendimento.

Em "*O Algoritmo NSGA-II Aplicado Ao Planejamento De Lavra De Minas A Céu Aberto*" de Guido Pantuza e Marcone Jamilson, apresentam uma adaptação do algoritmo evolutivo multiobjetivo NSGA II para o problema de planejamento de lavra em minas a céu aberto. Neste problema, cada frente possui características de qualidade diferentes e o ritmo de lavra deve ser realizado de forma proporcional, gerando uma alimentação que atenda a metas de qualidade e produção requeridas com um número reduzido de caminhões. Na adaptação feita, o NSGA II utilizou o Método de Descida em Vizinhança Variável como procedimento de busca local, o qual funcionou como operador de mutação[11].

Em "*Uma Plataforma em Software para Controle de Veículos Não Tripulados e Planejamento de Trajetórias*" outro trabalho, buscou-se analisar os mecanismos e meios de controle por meio de plataformas em software, que permitem interagir com drones de pequeno porte, de forma direta ou com um mínimo de interferência humana, em diferentes cenários de utilização. Possibilitando, assim, a execução de tarefas ou missões ditas autônomas com um mínimo de informação sobre o ambiente. Pode-se configurar completamente as ações e condições que propiciarão a tomada de decisões desses veículos, no sentido de evitarem colisões contra objetos próximos, desviarem e retornarem à trajetória da missão ou mesmo a abortarem. Utilizou-se algoritmo multiobjetivo NSGA-II, para o desenho de possíveis trajetórias. Tal abordagem é fundamentada no levantamento de dois objetivos: executar a missão consumindo um mínimo de recursos necessários ao voo e reduzir o risco de colisões contra obstáculos fixos dispostos ao longo do trajeto[12].

Diante dos trabalhos expostos, podemos observar que nenhum se aproxima do tema proposto neste projeto. Nosso trabalho possui algumas características que se destacam, como:

- Otimização multiobjetiva;
- Utilização de código aberto;
- Disponibilização de ferramenta para gerenciamento de situações emergenciais;
- Leitura dos dados de entrada padronizados em formato aberto (JSON);
- Disponibilização de dados de saída padronizados em formato aberto (JSON);
- Integração com a ferramenta *RESCUER*[1]*;*

## 4. Descrição do trabalho

Diante dos problemas expostos, decidimos criar uma solução computacional que ajude a diminuir os problemas de gerenciamento emergencial de lugares fechados. Assim, sugerindo uma ferramenta que gera soluções quais podem ser utilizados em sistemas computacionais a fim de salvar vidas.

Considerando o cenário-problema, a solução e todas fatos apresentados, nossa tentativa noa uso do algortimo é minimizar duas funções objetivo. A primeira função objetivo é a de balanceamento das saídas.

$$\sum_{\substack{bal \\ i = 1,n \\ j = 1,n \\ i \neq j}} \left| Si - Sj \right|$$

**Equação 1: Função de balanceamento.**

Esta função-objetivo nos garante um melhor balanceamento do número de pessoas que irão para as saídas. Seja *Si* o valor que representa o número de pessoas alocadas para a saída *i*, *Sj* representa a quantidade de pessoas que irão se dirigir à saída *j*. Os valores de *i* e j podem variar até *n.* O valor de *n* é o número máximo de saídas existentes. Quanto mais próximo de zero for o resultado desta função, mais eficiente será o balanceamento.

Similarmente ocorre com a função de tempo de fuga. Para chegarmos a um valor aceitável, utilizaremos a localização dos indivíduos, das saída e suas distâncias.

$$\sum_{\substack{dist \\ i = 1}}^{w} (i , q_{(i)})$$

**Equação 2: Função de tempo de fuga.**

A função-objetivo do tempo de fuga é o somatório da distância entre a posição de cada indivíduo e a localização da respectiva saída designada para este. O valor de *i* caracteriza a localização do indivíduo. O valor de *w* corresponde ao número total de pessoas do ambiente. A função $q_{(i)}$ representa a localização da saída designada para o indivíduo *i*, valor retirado do alelo correspondente à posição.

[1] RESCUER visa desenvolver uma plataforma de computador inteligente e interoperável para o uso de informações de crowdsourcing esmagadas com dados abertos para apoiar a gestão de emergência e crises.

Nossa proposta é construir um componente de software que receba representações JSON com as informações do ambiente e das pessoas nele contidas. Este realizará os devidos cálculos computacionais por meio do algoritmo genético buscando minimizar as funções objetivos apresentadas. Como saída, gerar outras representações JSON com as possíveis melhores rotas de fuga.

Os cromossomos são definidos de acordo com a quantidade de pessoas e quantidade de saídas existentes. A população inicial é gerada de forma aleatória (referimo-nos aos alelos), com um tamanho baseado em testes. A avaliação das soluções segue o critério determinado pelas funções que tentamos minimizar. Estas medem a qualidade do indivíduo (função de aptidão), baseadas no balanceamento das saídas e tempo de evacuação (menores distâncias percorridas). O método de seleção dos indivíduos para realização do cruzamento é o de seleção por torneio. Nos operadores genéticos, a técnica de cruzamento a ser aplicada é a de *Single-point Crossover* e a mutação, mudança aleatória do valor do alelo. As taxas de mutação e crossover foram definidas baseadas em experimentos e trabalhos relacionados.

## 4.1 JSON

Escolhemos este formato de padrão aberto, pois este utiliza texto legível a humanos para transmitir objetos de dados. Sua estrutura consiste em pares de atributo-valor. Possui algumas vantagens em relação aos outros formatos como:

- legibilidade
- independência de linguagem
- analisador (parsing) mais fácil de utilizar
- formato tipado
- suporta objetos; e outras.

Os JSONs de entrada possuem as informações do ambiente (dimensões, quantidade de subambientes, layout e quantidade de saídas) e das pessoas contidas nos subambientes (quantidade e localização destas).
Para uma possível integração com este componente computacional e utilização com outros cenários-problema, deve-se utilizar a seguinte tipagem dos dados e sintaxes:

```
[
        {"positionY":42.408621824318345,
         "id":0,"type":"emergencyExit",
         "positionX":47.37237342475748},
        {"positionY":43.884383784242125,
         "id":1,"type":"emergencyExit",
         "positionX":46.15054087810854},
        {"positionY":41.33011113753943,
         "id":2,"type":"emergencyExit",
         "positionX":40.68739438424117},
        {"positionY":41.131620155345004,
         "id":3,"type":"emergencyExit",
         "positionX":45.64112370733204},
        {"positionY":40.26551003484992,
         "id":4,"type":"emergencyExit",
         "positionX":40.63245986893338},
        {"positionY":48.65011361652005,
         "id":5,"type":"emergencyExit",
         "positionX":43.1352160930362}
]
```

**Figura 10: Exemplo de arquivo JSON com as definições do ambiente.**

```
[                           // definições das pessoas contidas nos ambientes
   {
      "idPerson":1,         // id - integer
      "positionX":7,        // posição x - float
      "positionY":1,        // posição y - float
      "stationary":false,   // dinamicidade - boolean
      "age":61,             // idade - integer
      "disability":1        // restrições - integer
   },
   {
      "idPerson":2,
      "positionX":4,
      "positionY":1.5,
      "stationary":false,
      "age":69,
      "disability":0
   },
   {
      "idPerson":3,
      "positionX":2,
      "positionY":5,
      "stationary":false,
      "age":25,
      "disability":0
   }
]
```

**Figura 11: Exemplo das definições das pessoas.**

A padronização do arquivo com as definições das saídas ficou da seguinte forma: [{ “id” : integer, “type” : “emergencyExit”, “positionX” : double , “positionY” : double }, …].

A Figura 11 contém a definição do arquivo com informações das pessoas contidas no ambiente.

## 4.2 Cromossomos

Para os cromossomos, adotaremos que o tamanho de cada cromossomo será equivalente à quantidade total de pessoas contidas no ambiente. A posição de cada alelo refere-se ao *id* de cada pessoa. O valor contido em cada seção do cromossomo, ou seja, valor contido em um alelo, será a numeração do identificador de cada possível saída.

Na solução do exemplo da figura à seguir (Figura 12), a pessoa com *id* 0 (referente à posição do alelo 0 no cromossomo) se dirigirá para a saída 2, assim como a pessoa com o id 1 também irá para a saída 2. Já a pessoa com o id 2, será direcionada para a saída 3, assim sucessivamente.

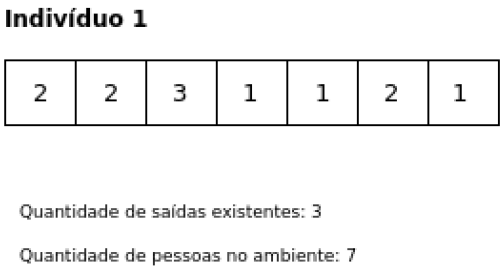


**Figura 12: Exemplo de um cromossomo nesta abordagem.**

## 4.3 População inicial

Utilizaremos tamanhos diferentes na população inicial. Para parametrização, utilizaremos valores aleatórios quais possam nos fornecer melhores soluções. Na fase de geração e análise dos dados, utilizaremos o melhor valor encontrado na fase de parametrização.

## 4.4 Avaliação da população

A avaliação da população, em cada geração, será realizada utilizando duas funções de aptidão, que indicarão a qualidade de cada indivíduo na população. Esta é a fase mais importante em qualquer algoritmo genético, pois através desta será medida quão

próximo um indivíduo está da solução desejada ou quão boa é esta solução. É essencial que a função que avalia a qualidade dos indivíduos seja muito representativa e diferencie na proporção correta as más soluções das boas. Se houver pouca precisão na avaliação, uma solução ótima pode ser posta de lado durante a execução do algoritmo, além de gastar um maior tempo computacional explorando soluções pouco promissoras [10].

Na função aptidão, para melhor pontuação do indivíduo, o valor do resultado de cada função aptidão deverá se aproximar de 1. O valor $f_{objetivo}$ possui os valores resultantes de cada função objetivo (balanceamento e fuga).

## 4.5 Seleção dos indivíduos para cruzamento

Para realizar a seleção dos indivíduos e transmitir a hereditariedade destes às populações seguintes, utilizaremos o método de seleção por torneio.

## 4.6 Cruzamento e taxa

Dentre muitas técnicas de recombinação, utilizaremos, no operador genético em questão, a técnica de cruzamento *Single-point Crossover.* Esta consiste em utilizar um único ponto aleatório (ponto de corte), no nosso caso, para recombinação em ambos os indivíduos progenitores. Todos os dados além dos pontos selecionados são trocados entre os progenitores.

Referente à taxa de cruzamento, percebe-se que quanto maior for, mais rapidamente novas estruturas serão introduzidas na população. Porém, se esta for muito alta, estruturas com boas aptidões poderão ser retiradas mais rapidamente, visto que a maior parte da população será substituída. Valores muito altos desta taxa podem estimular a perda de estruturas com altas aptidões. Com um valor muito baixo, o algoritmo pode se tornar excessivamente lento. É indicado utilizar inicialmente a taxa aleatória dentro do intervalo típico recomendado de $\rho_r \in [0{,}5;\ 1{,}0]$ para o cruzamento [10]. Diante destes fatores e grande referência na literatura, decidimos optar em utilizar a taxa de $tx_{cruzamento} = 0{,}8$ *e* $0{,}6$.

## 4.7 Mutação e taxa

Visando fornecer novas informações dentro da população e, desta forma, aumentar a diversidade populacional, utilizaremos um valor aleatório dentro do intervalo recomendado na literatura [10] especializada de $\rho_m \in [0,\ 01;\ 0{,}05]$.

## 4.8 Framework

Com o intuito de padronizar este trabalho, utilizamos um framework denominado JMetal. Este é construído na linguagem JAVA, orientado a objetos, ideal para otimização multiobjetiva com meta-heurísticas. Possui características como: portabilidade, indicadores de qualidade e representações variáveis, dentre outras.

Criamos as classes *PersonJSONCreator.java* e *EnvironmentJSONCreator.java* que geram arquivos com a estrutura descrita anteriormente, no formato JSON. Estes arquivos contem informações de cenários e pessoas neles contidos, possibilitando assim testes em diversos cenários. Os cenários descritos posteriormente foram gerados através destes geradores.

Para leitura e utilização das informações sobre as pessoas e os ambientes, foram criadas as classes: *PersonsJSONReader.java*, *EnvironmentJSONReader.java, Person*, *Exit*, *Chromosome*,.

Também criamos a classe *SeedReader.java*. Esta é responsável por ler o arquivo com um valor estático do *seed*. Com a manipulação do valor do *seed*, mantendo-o fixo, retiramos a aleatoriedade na fase de parametrização.

A classe *EscapeRoute.java* contém o algoritmo de criação e avaliação das soluções.

A classe *NSGAII_main.java* possui os parâmetros de configuração e os métodos de criação e manipulação dos objetos utilizados na construção do algoritmo.

Criamos uma classe *AnalizeSolutions.java, para análise estatística das soluções.*

As saídas foram gravadas em arquivos, dentro da pasta "*tests*" e das subpastas no formato "**X**persons_**Y**exits_cross**Z.Z**_mut**W.W**_population**K**_evolutions **M**_seedfixed**N**".

As melhores soluções foram selecionadas e convertidas para um arquivo no formato *JSON*. A referência "*chave : valor"* utilizada foi " *idPerson : idEmergencyExit "*. Este arquivo também se encontra dentro da pasta.

Todas as classes criadas encontram-se em um pacote separado dos pacotes do framework.

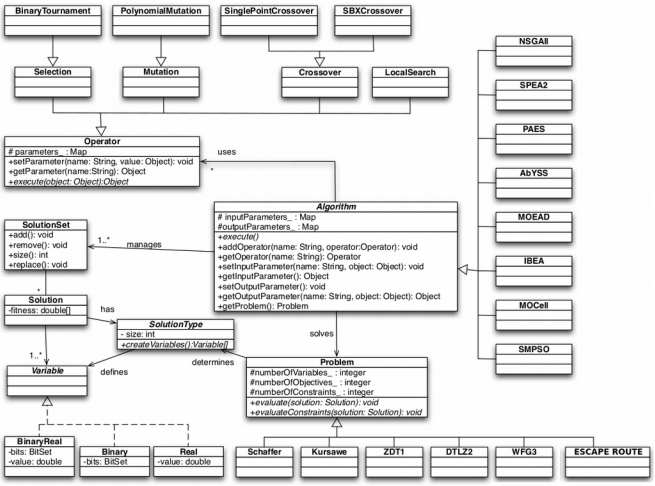


**Figura 13: Diagrama de classes.**

# 5. METODOLOGIA

Levamos em consideração que é difícil de definir uma única solução que supra todas as necessidades e não sabermos qual é a melhor solução, de forma simples e direta. Mas é possível verificar a geração de ótimas soluções com testes funcionais, com o recebimento dos valores de entrada, resultando em valores de saída.

Definimos a quantidade da população inicial aleatoriamente, mas dentro da faixa baseada na quantidade de saídas e de pessoas.

Os valores dos alelos das soluções da população inicial foram todos gerados de forma aleatória (utilizando o *seed* para a pseudo-aleatoriedade).

Uma das métricas para avaliação que utilizamos foi o nível de balanceamento. Esta permite equalizar a quantidade de pessoas em cada saída existente. A maior distância em relação ao valor

médio será definida como o índice desta métrica. Diante disto, após várias execuções do algoritmo, vimos que os valores gerados foram próximos dos índices, não havendo uma discrepância muito grande entre os valores. Mesmo alguns valores sendo considerados diferentes, estes, ainda assim, encontraram-se próximos do índice.

Em relação à segunda métrica utilizada, o tempo de fuga, utilizamos a correspondência em relação às distâncias. Mesmo havendo a variação em torno do índice.

Realizamos testes com doze configurações diferentes da taxa de cruzamento, taxa de mutação e gerações para um cenário-problema a fim de estabelecer parâmetros ideais. As características do cenário utilizado na parametrização foram as seguintes:

- Indivíduos: Cromossomo de tamanho 20000. Alelos variando entre 0 e 50;
- O valor de *seed* fixo em 0,5;
- O número máximo de evoluções foi fixado em 250;
- A técnica de cruzamento utilizada foi a *Single-point Crossover;*
- A técnica de mutação utilizada foi a *BitFlipMutation;*
- A técnica de seleção utilizada foi a *BinaryTournament2*, característica do NSGAII;
- Os cenários possuem as dimensões de 500 x 500;

Os demais parâmetros seguiram a tabela a seguir:

| Cruzamento | Mutação | População |
|---|---|---|
| 0.8 | 0.01 | 100<br>200<br>500 |
| 0.8 | 0.02 | 100<br>200<br>500 |
| 0.6 | 0.01 | 100<br>200<br>500 |
| 0.6 | 0.02 | 100<br>200<br>500 |

**Tabela 01 : Descrição das configurações.**

Para definição dos valores de média de balanceamento, utilizamos a média aritmética.

Como medida de dispersão, utilizamos o cálculo do desvio padrão populacional.

No somatório do balanceamento, foram consideradas as quantidades de ocorrências em cada saída.

A distância total percorrida é caracterizada pela soma de todas as distâncias percorridas pelas pessoa até sua saída respectiva.

Após a coleta dos dados das soluções ótimas geradas, mapeamos estes dados e conseguimos informações similares a da configuração um:

Configuração 1

| | f1(balanceamento) | f2(distancia) |
|---|---|---|
| Cromossomo 0 | 45980.0 | 7435955.917194824 |
| Cromossomo 1 | 54976.0 | 7430904.517223019 |
| Cromossomo 2 | 43924.0 | 7444367.194214746 |
| Cromossomo 3 | 56204.0 | 7428528.983118201 |
| Cromossomo 4 | 42696.0 | 7477398.883553377 |

**Tabela 02: Dados resultantes das soluções do cenário – configuração 1.**

Os demais dados de parametrização encontram-se em anexo na tabela “Configurações para parametrização”.

Após calcularmos as médias para os valores de balanceamento e distâncias totais de cada configuração, conseguimos encontrar os melhores valores da taxa de cruzamento e de mutação na configuração 2. Sendo, $Tx_{Cruzamento}$ = 0,8 e $Tx_{Mutação}$ = 0,01. Já a população inicial, igual a 200. Estes valores são os respectivos valores da configuração 2, a qual obteve o menor valor na média de balanceamento e um valor aceitável na média das distâncias totais percorridas. Conforme pode ser observado na Tabela 03, fora realizado uma classificação no dados, obedecendo a ordem crescente.

Vale salientar que em todos os cenários os valores ficaram muito próximos.

| | MédiasBalanceamento | DistânciasTotais |
|---|---|---|
| Configuração 2 | 43740,8 | 7452604,88970042 |
| Configuração 11 | 44144 | 7448006,61374603 |
| Configuração 8 | 44386,4 | 7448578,85060579 |
| Configuração 5 | 44454 | 7433034,74287281 |
| Configuração 4 | 47303,6 | 7451866,18282146 |
| Configuração 3 | 47745 | 7440364,06309368 |
| Configuração 6 | 47745 | 7440364,06309368 |
| Configuração 9 | 47745 | 7440364,06309368 |
| Configuração 12 | 47745 | 7440364,06309368 |
| Configuração 10 | 48397 | 7440789,95841573 |
| Configuração 7 | 48710,4 | 7447659,38143772 |
| Configuração 1 | 48756 | 7443431,09906083 |

**Tabela 03: Dados resultantes das médias aritméticas das soluções dos cenários.**

Após validarmos a melhor configuração das taxas, população inicial e técnicas de cruzamento e mutação, realizamos três execuções do algoritmo nos cenários descritos na Tabela 04. Nosso objetivo é gerar dados de soluções ótimas. Esses dados serão utilizados em testes estatísticos. O resultado dos testes estatísticos nos orientaram quanto a validação da nossa hipótese.

Seguem informações dos cenários:

| Pessoas | Saídas | Seed |
|---|---|---|
| 500 | 10 | Aleatório |
| 1000 | 15 | Aleatório |
| 5000 | 20 | Aleatório |
| 10000 | 25 | Aleatório |
| 20000 | 50 | Aleatório |

**Tabela 04 : Descrição dos cenários-problema principais.**

Nestes cenários-problema utilizados para validação do algoritmo, utilizamos a configuração citada anteriormente, a que mais conseguiu minimizar as funções-objetivo. As demais características foram:

- Número máximo de gerações: 250;
- Cruzamento: *Single-point Crossover;*
- *Mutação: BitFlipMutation;*
- *Seleção: BinaryTournament2;*
- *Dimensões dos cenários: 500 x 500;*

Como demosntração, podemos apresentar os resultados do cenário 5 em sua segunda execução:

Cenário 5B

| | f1(balanceamento) | f2(distancia) |
|---|---|---|
| Cromossomo 0 | 45228.0 | 7439030.321418173 |
| Cromossomo 1 | 36344.0 | 7475161.965293214 |
| Cromossomo 2 | 48564.0 | 7434530.924307485 |
| Cromossomo 3 | 52928.0 | 7430361.881307769 |
| Cromossomo 4 | 63692.0 | 7429223.343404917 |
| Cromossomo 5 | 42164.0 | 7457295.920003165 |
| Cromossomo 6 | 50804.0 | 7433286.197385751 |
| Cromossomo 7 | 37844.0 | 7470702.327246913 |

**Tabela 05: Dados das soluções do cenário 5, em sua segunda execução.**

Podemos observar na Figura 14 que as soluções geradas possuem comportamento similar independente do *seed* utilizado. Esta figura ilustra as três execuções consecutivas do algoritmo sobre o cenário 5.

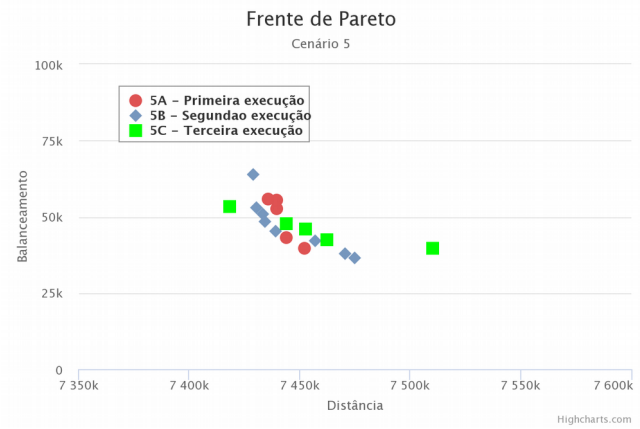


**Figura 14: Gráfico com distribuição das soluções do cenário 5.**

Os demais dados das soluções ótimas geradas a partir dos cenários citados na Tabela 03 e com a melhor configuração, encontram-se em anexo na tabela “Dados das soluções para validação”.

As soluções da frente de Pareto geradas possuem a quantidade de cromossomos diferente umas das outras (até nas soluções do mesmo cenário-problema). Portanto, para validarmos a nossa hipótese e diante das amostras disponíveis, utilizamos um método estatístico denominado Teste de Kruskall Wallis.

O teste de Kruskal Wallis foi escolhido, pois, é um método não-paramétrico usado para testar se um conjunto de amostras provém da mesma distribuição. Ele é usado para testar a hipótese nula de que todas as populações possuem funções de distribuição iguais contra a hipótese alternativa de que ao menos duas das populações possuem funções de distribuição diferentes.

A fim de evitar correções trabalhosas e possíveis erros, melhorando o processo de validação do teste estatístico, utilizamos o R. O ambiente de desenvolvimento integrado para cálculos estatísticos e gráficos, R, foi utilizado em conjunto com o RStudio. Este segundo é um ambiente de desenvolvimento integrado (IDE) para R que inclui um console, sugestão de sintaxe e editor que suporta execução de código direto, bem como ferramentas para plotagem, histórico, depuração e gerenciamento do espaço de trabalho.

Para aplicarmos o teste de Kruskal Wallis no RStudio, sobre as amostras das soluções, classificamos os cromossomos das soluções da frente de Pareto por grupos. O conjunto de soluções resultantes de cada execução do algoritmo sobre um cenário ficou designado a um grupo.

| | f1(balanceamento) | f2(distancia) | Grupo |
|---|---|---|---|
| 1 | 304.0 | 188659.20693770706 | 1 |
| 2 | 320.0 | 185615.3049769928 | 1 |
| 3 | 408.0 | 182569.47478721186 | 1 |
| 4 | 584.0 | 179751.09844968855 | 1 |
| 5 | 536.0 | 182273.07924990886 | 1 |
| 6 | 552.0 | 180923.76801508878 | 2 |
| 7 | 616.0 | 178689.51470469238 | 2 |
| 8 | 480.0 | 182805.45764611705 | 2 |
| 9 | 336.0 | 183454.5463724601 | 2 |
| 10 | 136.0 | 186169.1874059489 | 2 |
| 11 | 416.0 | 182883.32164161289 | 3 |
| 12 | 320.0 | 184385.9595950069 | 3 |
| 13 | 800.0 | 180602.22052622613 | 3 |
| 14 | 456.0 | 184104.9477463355 | 3 |
| 15 | 488.0 | 180803.27960859734 | 3 |
| 16 | 972.0 | 179947.57087367497 | 3 |
| 17 | 460.0 | 183153.2230640376 | 3 |
| 18 | 212.0 | 187481.30429288195 | 3 |

**Tabela 06: Dados agrupados do cenário 1 para aplicação do teste estatístico.**

Após os cálculos estatísticos realizados, obtivemos os seguintes resultados para cada função objetivo de cada cenário:

```
Kruskal-Wallis rank sum test

data:  f1.balanceamento. by Grupo
Kruskal-Wallis chi-squared = 0.26628, df = 2, p-value = 0.8753
```

**Figura 15: Resultado do teste de Kruskal Wallis para a função de balanceamento do cenário 1.**

```
        Kruskal-Wallis rank sum test

data:  f2.distancia. by Grupo
Kruskal-Wallis chi-squared = 0.13421, df = 2, p-value = 0.9351
```

**Figura 16: Resultado do teste de Kruskal Wallis para a função de distância do cenário 1.**

Para validarmos este trabalho, buscamos provar a hipótese nula. Esta consiste em afirmar que os parâmetros ou características matemáticas de duas ou mais populações são estatisticamente idênticos. Assim, utilizamos o *p-value* resultante do teste estatístico para confrontar a hipótese nula.

O *p-value* é a medida que fornece a probabilidade de observar em um teste estatístico, no mínimo como extremo, o valor realmente observado, assumindo que a hipótese nula seja verdadeira. O p-value depende diretamente de uma dada amostra. Este tenta fornecer uma medida para a força dos resultados de um teste. Se a hipótese nula for verdadeira e a chance da variação aleatória for a única razão para as diferenças amostrais, então o p-*value* é uma medida quantitativa para alimentar o processo de tomada de decisão como evidência.

Observamos a tabela de interpretação do *p-value*:

| P-value | Interpretação |
|---|---|
| $p < 0,01$ | evidência muito forte contra $H_0$ . |
| $0,01 <= p < 0,05$ | evidência moderada contra $H_0$. |
| $0,05 <= p < 0,10$ | evidência sugestiva contra $H_0$. |
| $0,10 <= p$ | pouca ou nenhuma evidência real contra $H_0$. |

**Tabela 07: Interpretações do *p-value*.**

Diante disto, observamos o sumário dos *p-values* resultantes dos testes de Kruskal Wallis dos cenários-problema com a configuração ideal:

| Cenário | P-value(balanceamento) | P-value(distância) |
|---|---|---|
| 1 | 0.8753 | 0.9351 |
| 2 | 0.5998 | 0.6412 |
| 3 | 0.2337 | 0.4066 |
| 4 | 0.9154 | 0.2789 |
| 5 | 0.7038 | 0.6282 |

**Tabela 08: Sumário dos *p-values* dos cenários.**

Assim, para determinar se algumas das diferenças entre as medianas são estatisticamente significativas, comparamos os valores dos p-values com o nível de significância para avaliar a hipótese nula. A hipótese nula afirma que estatisticamente as medianas da população são todas iguais. Utilizamos o nível de significância padrão da bibliografia (denotado como α ou alfa) de 0,05. Um nível de significância de 0,05 indica um risco de 5% de concluir que existe uma diferença quando não há diferença real.

Podemos ressaltar que houveram diferenças entre os resultados do teste de Kruskal Wallis dos cenários, pois os valores de entrada (seeds, cenários, populações, dentre alguns) que geraram os dados das soluções analisadas (soluções Pareto-ótimas) são aleatórios. Consequentemente, os valores de *p-value* dos cenários são naturalmente diferentes entre sí. Esta diferença do *p-value* entre os cenários, não invalida a afirmação da hipótese nula. Acreditamos que os valores dos *p-values* dos cenários 3 (balanceamento) e 4 (distância) se mostraram um pouco discrepantes dos demais devido à pequena quantidade de soluções ótimas geradas (amostras utilizadas nos testes estatísticos).

As diferenças entre as medianas não são, estatisticamente, significativas entre si. Como o valor de *p-value* em cada cenário foi maior que o nível de significância, não temos evidências suficientes para rejeitar a hipótese nula. Na maioria dos casos, existe alta evidência de que as amostras são da mesma população.

# 6. Conclusão

Neste trabalho foi aplicado o algoritmo genético NSGA-II em cenários-problema com situações emergenciais a fim de gerar rotas de fuga. As duas funções-objetivo minimizadas foram as distâncias das rotas (pessoa - saída) e o balanceamento das saídas (número de pessoas em cada saída).

Realizamos estudos para definições matemáticas das funções objetivo, dos parâmetros iniciais, estrutura e extensão dos arquivos de entrada e saída, método de cruzamento e mutação e suas respectivas taxas, valores de *seeds*, populações iniciais e características dos cenários.

Utilizamos o framework *JMetal* como base para desenvolvimento do trabalho. Este é escrito na linguagem *JAVA*.

Para definição da melhor configuração do algoritmo, foram realizados testes em um mesmo cenário com parâmetros diferentes, mas *seed* fixo, com a ideia de encontrar a configuração que melhor minimizasse as duas funções-objetivo para o tipo de problema-cenário proposto.

Também foram criadas classes utilitárias no framework para facilitar o uso, padronizar os métodos e minimizar os erros durante os testes. Estas classes foram utilizadas para manipular os arquivos JSON de entrada (com informações dos cenários e das pessoas contidas nestes), separar os testes em pastas e arquivos específicos, gerar arquivos JSONs com as soluções ótimas e demais informações de cada execução e analisar os dados gerados.

Após encontrarmos a melhor configuração, executamos o algoritmo, algumas vezes, em cenários diferentes e levantamos dados sobre as soluções. Em posse destes dados, realizamos testes estatísticos com os conjuntos-solução de cada cenário para comprovar a hipótese.

Com todas as informações contidas neste trabalho, não encontramos evidências que rejeite a hipótese nula. Acreditamos que, devido aos valores de p-value dos testes estatísticos serem satisfatórios na grande maioria dos cenários, possuímos indicações de que o algoritmo fornece soluções que convergem para uma mesma distribuição estatística. Mesmo obtendo valores de p-value menos satisfatórios em poucos casos, estes justificam-se devido à pequena quantidade de amostras analisadas desses cenários.

Podemos deduzir que, com uso da melhor configuração do algoritmo, soluções que convergem para valores ótimos serão fornecidos.

## 6.1 Trabalhos futuros

Podemos citar algumas melhorias que podem ser utilizadas para a elaboração de trabalhos futuros:

- Considerar as diferenças de locomoção das pessoas. Cadeirantes, idosos e crianças possuem velocidades de locomoção diferentes, influenciando no tempo de fuga, por exemplo.
- Avaliar a diferença no tempo de evacuação das pessoas considerando paredes, divisões, subambientes, móveis e demais objetos do local.

- Definir e aplicar no algoritmo o tamanho (comprimento) das saídas. Isto pode afetar diretamente na quantidade de pessoas que podem sair ao mesmo tempo em cada saída emergencial.
- Considerar mudanças repentinas no ambiente ou no processo de evacuação das pessoas. Isto exigirá uma nova execução do algoritmo com os novos dados do ambiente e das pessoas contidas nele de forma automatizada.

## 7. REFERÊNCIAS


[1] K. Deb ; A. Pratap ; S. Agarwal ; T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II, 07 August 2002.

[2] CASTRO, R. E. Otimização de estruturas, com multiobjetivos via algoritmo genético de pareto, UFRJ - Rio de Janeiro, 2001.

[3] L. Narciso, A. Maslinkiewicz, and D. R. J. de Freitas. Levantamento de Doenças Respiratórias e sua Associação com Ambientes Climatizados na Comunidade da Universidade do Oeste de Santa Catarina (UNOESC) de Xanxerê. Unoesc & Ciência - ACBS, 2014.

[4] C. A. P. Lugon. Curso de Formação de Bombeiro Profissional Civil - Gerenciamento de Emergências. Centro de Ensino e Instrução de Bombeiros - Seçao de Cursos de Extensão, Serra-ES .

[5] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. Evolutionary Algorithms for solving multiobjective problems, chapter 1, page 11. GENETIC AND EVOLUTIONARY COMPUTATION. Springer, 2 ed., 2002.

[6] J.H. Holland. Adaption in natural and artificial systems. MI: University of Michigan Press, 1975.

[7] Érico de Oliveira Costa Zini, Algoritmo Genético Especializado na Resolução de Problemas com Variáveis Contínuas e Altamente Restritos, Unesp – SP, 2009.

[8] ZINI, E. O. C. Introdução à programação linear. UFMS, Três Lagoas, 2005.

[9] POZO, A.; CAVALHEIRO A. F.; ISHIDA, C.; SPINOSA, E.; RODRIGUES E. M. - Computação evolutiva. Universidade Federal do Paraná, 61p. (Grupo de Pesquisas em Computação Evolutiva, Departamento de Informática- Universidade Federal do Paraná).

[10] ROMERO, R. Planejamento de sistemas de transmissão usando algoritmo genético. Ilha Solteira: UNESP, 2005.

[11] Guido Pantuza Júnior, Marcone Jamilson Freitas Souza. O Algoritmo Nsga-II Aplicado Ao Planejamento De Lavra De Minas A Céu Aberto. Universidade Federal de Ouro Preto. Campus Universitário Morro do Cruzeiro – Ouro Preto, MG. Rio de Janeiro- Brasil, 12 e 13 de agosto de 2010.

[12] Alberto Rogério e Silva. HEALTHDRONES – Uma Plataforma em Software para Controle de Veículos Não Tripulados e Planejamento de Trajetórias. Universidade Federal Rural de Pernambuco - Departamento de Estatística e Informática Programa de Pós-Graduação em Informática Aplicada. Setembro de 2016.

[13] Renato Lima Novais, Manoel G. Neto, Eduardo Santana, Karina Villela, Rita Suzana Maciell, Vaninha Vieira dos Santos, Adolfo Almeida Duran, Hans Dieter Rombach, Agma Juci Machado, Gustavo Perez, Rodrigo Silveira, Paul Lukowicz, Ralph Carbon - RESCUER: Reliable and Smart Analysis of Crowdsourcing Information for Emergency and Crisis Management. IFBA (GSORT – Grupo de Pesquisa em Sistemas Distribuídos, Otimização, Redes e Tempo Real), UFBA, 2014 – Atual.