UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JÔNATAS TRABUCO BELOTTI

MODELO DE ALOCAÇÃO DE FLUXO EM REDES PARA EVACUAÇÃO DE POPULAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2015

JÔNATAS TRABUCO BELOTTI

MODELO DE ALOCAÇÃO DE FLUXO EM REDES PARA EVACUAÇÃO DE POPULAÇÃO

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientadora: Prof.ª Dr.ª Sheila Morais de Almeida

PONTA GROSSA
2015

RESUMO

O presente trabalho apresenta um Modelo de Alocação de Fluxo capaz de determinar quais as rotas de fuga para populações em áreas de risco. Tal evacuação pode ser realizada de duas formas: levar a população para abrigos predeterminados ou apenas retirar a população da área de risco. A abordagem utilizada é a relaxação do problema de evacuação de população em áreas de risco, para o qual resulta no conhecido Problema do Fluxo Máximo. A partir da solução do problema relaxado, os esforços serão para incluir no modelo final a maioria das restrições ignoradas, garantindo que tal modelo possa ser resolvido eficientemente. Além disso, o desenvolvimento deste projeto visa contribuir para aumentar a velocidade de tomada de decisão em situações reais, onde o principal objetivo é o salvamento de vidas.

Palavras-chaves: Fluxo em redes, rotas de fuga, fluxo máximo, fluxo com múltiplas origens e destinos.

ABSTRACT

This study aims to develop a flow Allocation Model to determine the escape routes for people in risk areas. The evacuation of populations can be accomplished in two ways: taking the population to predetermined safe places or only remove the population from the areas of risk. In the literature flow problems can be easily found and there are many efficient algorithms to solve it. An initial approach is to solve a relaxation of evacuation of population in risk areas which results in the known Maximum Flow Problem. Given the solution of the relaxed problem, the efforts will be to include in the final model most of restrictions ignored, esuring that this model afford an efficient solution. In addition, the objective of this project is to contribute to increase the speed of the decision-making in real situations, where the primary goal is saving lives.

Key-words: Network flow , escape routes , flow with multiple sources and destinations.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Diversas catástrofes marcam a história da humanidade, tomando como exemplos o terremoto em Aleppo na Síria ocorrido em 1138 (VIRGULA, 2010), o ciclone Bhola em 1970 (HURRICANES, 2005), o tufão Heiyan nas Filipinas em 2013 (UOL, 2013) e as enchentes na Europa Central em 2013 (BBC-BRASIL, 2013). Nessas situações, centenas ou milhares de pessoas morreram ou ficaram feridas. A existência de um plano de evacuação que permita retirar todas ou pelo menos a maioria das pessoas de um local de risco é um serviço de utilidade pública que pode salvar muitas vidas.

O tema deste trabalho é uma etapa do plano de evacuação, considerando um local onde há o risco de ocorreu uma tragédia e o número de pessoas nesse local, determinar quais as rotas que a população deve tomar para chegar em segurança e o mais rápido possível aos locais seguros, que chamamos de abrigos.

Em sua tese de doutorado, Campos (1997) afirma que as principais restrições para a realização da evacuação da população de regiões de risco estão relacionadas com a grande quantidade de indivíduos, a capacidade limitada das vias e o tempo escasso para a realização da operação. Quanto maior é o número de restrições impostas e maior é a quantidade de indivíduos e vias que precisam ser consideradas, mais elevado é o custo computacional dos algoritmos projetados para resolver o problema.

Do ponto de vista computacional, quanto maior o número de restrições envolvidas no problema, mais custoso é encontrar um plano de evacuação adequado. O presente trabalho apresenta um método que mostrou-se eficiente para determinação de rotas de evacuação de população em situações de risco.

Na próxima sessão são apresentados os objetivos que norteiam o desenvolvimento deste trabalho. No Capítulo 2 apresenta a descrição completa do problema abordado juntamente com todos as restrições impostas para a solução. O Capítulo 3 trás todos os conceitos teóricos necessários para o entendimento completo do trabalho, juntamente com os trabalhos presentes na literatura que se relacionam com o tema abordado. No Capítulo 4 é apresentado o método proposto para solução do problema. Os testes realizados com o modelo proposto são descritos e apresentados no Capítulo 5. O Capítulo 6 apresenta os pontos que podem ser melhorados futuramente neste trabalho. Por fim as conclusões a respeito do método são apresentadas no Capítulo 7.

1.1 OBJETIVOS

Esta seção apresenta o objetivo geral e os objetivos específicos considerados durante o desenvolvimento do trabalho.

1.1.1 Objetivo Geral

Desenvolver um modelo de alocação de fluxo aplicado na escolha das rotas de fuga para uma população em áreas de risco.

1.1.2 Objetivos Específicos

Para alcançar o objetivo principal deste trabalho, alguns objetivos específicos foram tomados como meta:

- Aprofundar os conhecimentos em grafos, mais especificamente em problemas de fluxo em redes.
- Delimitar o problema específico a ser trabalhado no contexto de planos de evacuação.
- Criar um modelo de solução para o problema delimitado.
- Testar o modelo proposto para o problema.

1.2 JUSTIFICATIVA

Um conjunto de rotas disjuntas é um conjunto de caminhos que não compartilham vértices. Rotas disjuntas podem facilitar o trânsito pelas vias e diminuir as chances de conflito que podem atrapalhar o fluxo e ser agravadas em situações de pânico. Entretanto o uso de rotas disjuntas exclui vias que poderiam estar presentes na solução ótima do problema. Na literatura encontram-se algoritmos para determinar rotas de fuga, como em Hutchinson (1979), Martin e Manheim (1965) e Papacostas e Prevedouros (1993).

Em Campos (1997) a autora propõe um método para determinar um plano de evacuação onde as rotas de fuga são disjuntas. A imposição de rotas totalmente disjuntas pode impedir que toda a população seja retirada das áreas de risco em tempo hábil, visto que diminui o número de vias disponíveis para uso concomitante. Neste trabalho houve uma relaxação do Problema de Evacuação de População, removendo-se a restrição de que as rotas devem ser disjuntas. O modelo proposto considera a possibilidade de existência de rotas que compartilham parcialmente o caminho e impõe algumas restrições em relação à orientação do fluxo e a quais interseções entre as rotas podem ser admitidas de forma a minimizar a possibilidade de conflito nas vias.

Como apresentado na Introdução, catástrofes são responsáveis por matar milhares de pessoas. A elaboração de modelos de definição de rotas de fuga juntamente com sua implementação tem aplicação prática no salvamento de vidas. Apesar disso existem poucos modelos para

tal. Assim sendo a elaboração desse modelo para definição de rotas de fuga para uma população em situação de catástrofe é uma contribuição relevante do ponto de vista prático, sendo um serviço de utilidade pública.

2 O PROBLEMA

O mapa viário de uma região (bairro, cidade ou país) pode ser modelado como um grafo, onde cada vértice representa um local e cada aresta representa a via (nesse caso rodovia, estrada ou rua) que faz a ligação entre os locais. A Figura 1 apresenta um exemplo de modelagem do mapa de um bairro através de um grafo.

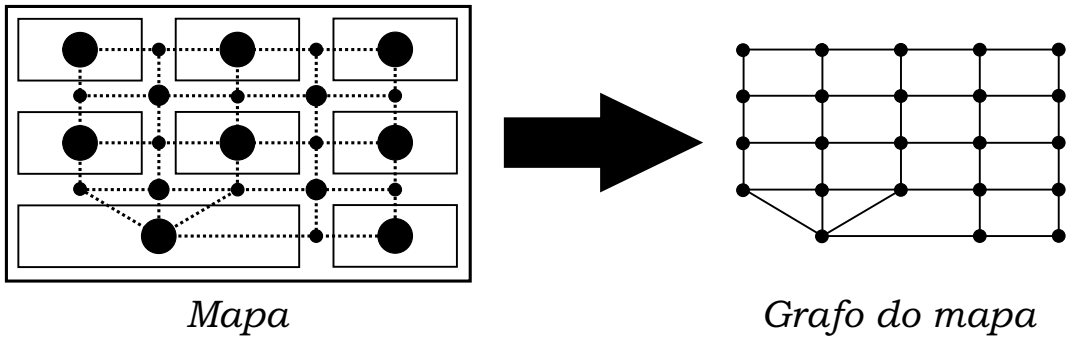

Figura 1 – Modelagem de mapa em grafo.

Fonte: Autoria própria

Cada via tem uma capacidade máxima de escoamento (no caso de rodovias, carros por hora). Assim deslocar pessoas por uma região é como estabelecer um fluxo em um grafo. A seguir são apresentadas algumas definições importantes para a compreensão deste trabalho, assim como a definição do problema abordado.

Dada uma quantidade de veículos que devem sair de uma ou mais regiões em risco de catástrofe e chegar às regiões seguras (saindo de um ou mais vértices e chegando em outros vértices), o fluxo é a quantidade de indivíduos por unidade de tempo (ex. carros por hora) que se deslocam pelo mapa em direção às regiões seguras. Dessa forma busca-se a maneira mais eficiente de escoar esse fluxo pela malha viária (grafo), atendendo às restrições de capacidade de cada via e de tempo hábil para realização do escoamento. Campos (1997) afirma que esse escoamento é mais eficiente quanto maior for a quantidade de indivíduos que conseguem alcançar locais seguros no menor tempo possível.

Teoricamente, problemas onde se deseja maximizar a quantidade de fluxo em um grafo são conhecidos como Problemas de Fluxo Máximo. Segundo Cormen *et al.* (2002) existem inúmeras variações desses problemas, por meio da imposição de limites na capacidade das arestas, do estabelecimento de custo por unidade de fluxo em cada aresta, do estabelecimento de fluxo mínimo em cada aresta, da imposição de consumo de parte do fluxo pelos vértices, etc. A Seção 3.1.3 apresenta o Problema de Fluxo Máximo.

No contexto desse projeto, deseja-se saber qual o maior fluxo que se pode escoar em uma rede que possui arcos com capacidade limitada, onde as rotas de fuga apresentam características bem definidas e com múltiplas origens e destinos.

2.1 ESCOAMENTO DE FLUXO

O que significa a medida de fluxo de uma rede? O fluxo é tratado como a quantidade do material a ser escoado que pode ser transportada pela rede em uma unidade de tempo. Tomando como exemplo uma rede de canos de água onde em 3 horas se bombeia 2000 litros d'água, o fluxo dessa rede é $\frac{2000}{3} = 666,66$ litros d'água por hora. Observe que, nesse exemplo, a unidade de tempo é hora.

Analisando detalhadamente, o fluxo é medido em um instante de tempo onde a rede se encontra totalmente preenchida. Observe que em um primeiro instante a rede se encontra vazia e o seu preenchimento pelo material demanda um certo tempo. No problema de escoamento da população em situações de catástrofes esse tempo deve ser levado em conta.

Dada uma população que reside em áreas de risco, abrigos para onde essas pessoas devem ser levadas e ruas por onde essas pessoas podem passar para chegar aos abrigos, quanto tempo demora para que essas pessoas alcancem os abrigos? Responder essa pergunta demanda mais informação a respeito da situação, como quantas pessoas devem ser retiradas, quantas pessoas cabem em cada abrigo, quanto tempo demora para uma pessoa percorrer cada via e quantas pessoas podem passar por cada via simultaneamente.

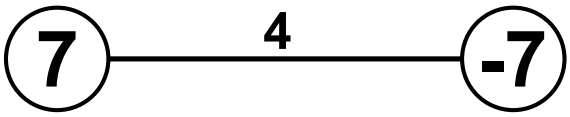

Figura 2 – Grafo G_1.

Fonte: Autoria própria

A Figura 2 apresenta o Grafo G_1, que fornece algumas dessas informações. O vértice a esquerda representa o local de onde as pessoas devem ser retiradas e o peso nesse vértice representa que 7 pessoas devem ser retiradas desse local. O vértice a direita representa o local para onde elas devem ser levadas e o peso negativo nesse vértice representa que 7 pessoas podem ser abrigadas nesse local. A aresta entre os vértices representa a via e a capacidade da aresta é de 4, ou seja a cada unidade de tempo 4 pessoas passam por essa via.

A capacidade representa a quantidade de indivíduos que podem trafegar na via em uma unidade de tempo, ou seja, quantos indivíduos a via comporta. A Figura 3 mostra que a aresta do problema apresentado suporta 4 pessoas trafegando por ela em um instante de tempo, logo sua capacidade é 4.

Desse modelo, pode-se obter as seguintes informações: a cada unidade de tempo, quatro pessoas passam pela via. Como há sete pessoas para serem retiradas do local de risco, e a via suporta 4 pessoas por unidade de tempo, serão necessárias $\lceil \frac{7}{4} \rceil = 2$ unidades de tempo para evacuação total dessa área.

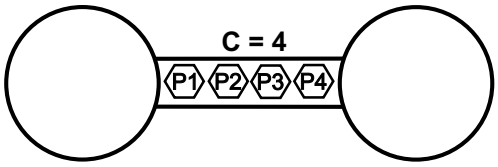

Figura 3 – Capacidade da via.

Fonte: Autoria própria

Entretanto essa interpretação do problema é pouco precisa. Observe que a capacidade da via significa que, no momento em que a via estiver operando na sua capacidade máxima, ou seja no momento em que a via estiver totalmente preenchida, sua vazão será de 4 pessoas por unidade de tempo. No problema considerado, é necessário levar em conta que nos momentos iniciais e nos momentos finais da evacuação as vias não estão totalmente preenchidas, visto que, nos momentos iniciais as primeiras pessoas estão entrando na via e nos momentos finais apenas os últimos a entrarem na via ainda não chegaram ao abrigo.

A Figura 4 mostra uma simulação do processo de locomoção das pessoas na via detalhando onde está cada pessoa em cada *frame*, até que todos cheguem ao abrigo. Observando a Figura 2.1 nota-se que a cada 4 *frames* 4 pessoas saem da via (*fames* 5, 6, 7 e 8). A capacidade da via representa a vazão da via por unidade de tempo quando a mesma estiver totalmente preenchida. Então, vamos considerar que uma unidade de tempo equivale a quatro *frames*, nesse exemplo.

O *frame* f_0 é o *frame* inicial, nesse momento todas as pessoas ainda estão na área de risco. Logo em seguida, no *frame* f_1, a primeira pessoa entra na via e começa a andar em direção ao abrigo. No *frame* f_2 mais uma pessoa entra na via. Em f_3 uma terceira pessoa entra na via. A partir do *frame* f_4 a via passa a operar em sua capacidade máxima, a partir desse momento tem-se 4 pessoas passando pela via, entretanto até agora nenhuma pessoa chegou ao abrigo. Por fim, no *frame* f_5, a primeira pessoa que entrou na via chega ao abrigo. Então, em virtude da capacidade da via, a partir do *frame* f_5, a cada unidade de tempo 4 pessoas chegam ao abrigo. Nos *frames* de f_4 até f_7 a via opera em sua capacidade máxima. A partir do *frame* f_8 a quantidade de pessoas na via diminui, pois não existem mais pessoas na área de risco. Dessa forma ninguém mais entra na via. Por fim, no *frame* f_{11} temos a evacuação completa, assim a evacuação durou 11 *frames*.

Sabemos pela capacidade da via, que em cada unidade de tempo 4 pessoas passam pela mesma. Na simulação, observa-se que a partir de f_5 em cada *frame* uma pessoa passa pela via, logo para que 4 pessoas passem pela via são necessários 4 *frames* , ou seja, 1 (uma) unidade de tempo. Dessa forma a simulação demorou $\frac{11}{4} = 2,75$ unidades de tempo. Como a unidade de tempo é uma medida inteira arrendondamos o valor para o menor inteiro maior que $2,75$. Então, a retirada das pessoas levou 3 unidades de tempo.

Infere-se então, que o tempo decorrido para transportar a população P de um local até o

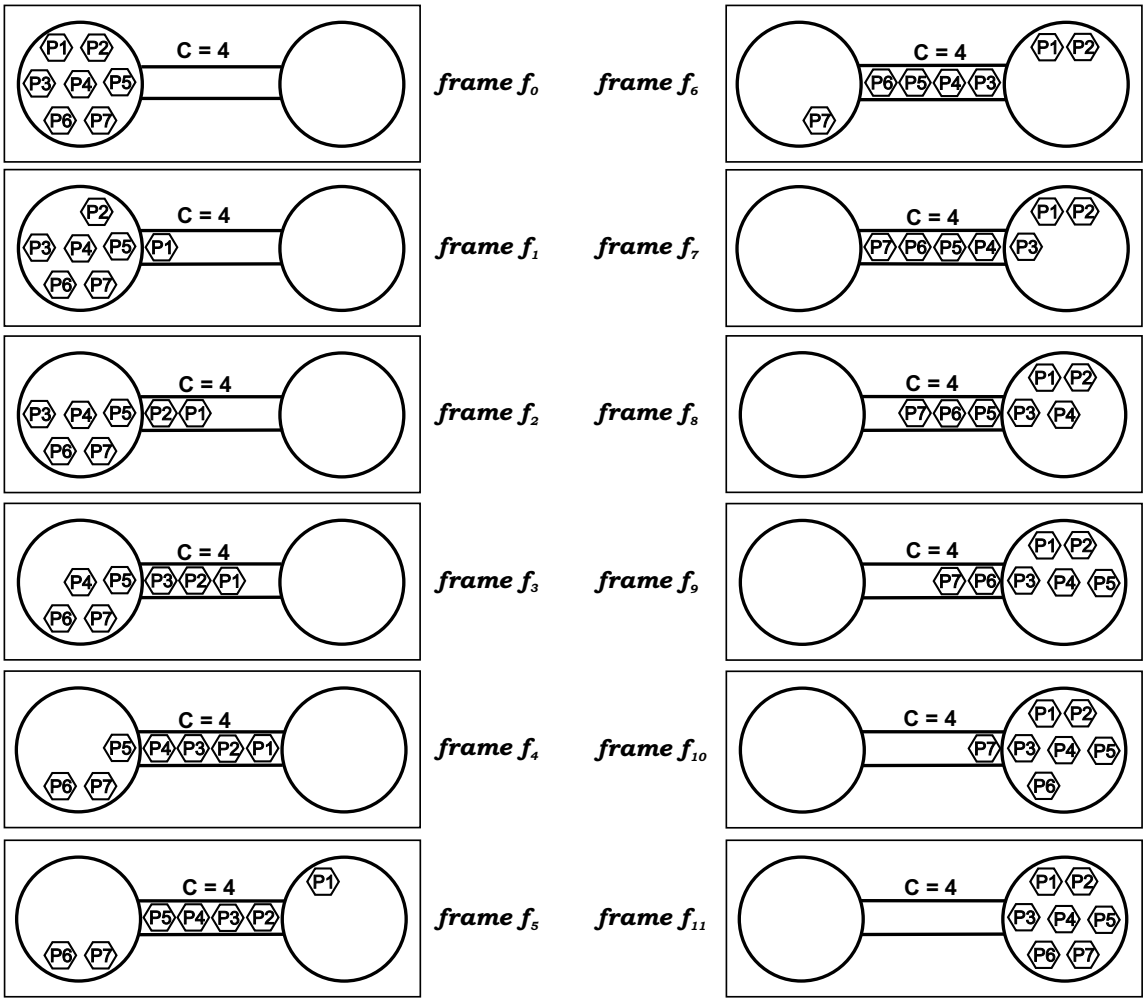

Figura 4 – Simulação de evacuação.

Fonte: Autoria própria

outro é determinado por $t = \lceil \frac{|P|}{c} \rceil + 1$, onde c é a capacidade da via, observando-se uma unidade de tempo extra necessária para o completo preenchimento da via, ou seja, o tempo até o primeiro indivíduo chegar ao abrigo. No exemplo apresentado temos $t = \lceil \frac{7}{4} \rceil + 1 = \lceil 1,75 \rceil + 1 = 2 + 1 = 3$ unidades de tempo.

2.2 RESTRIÇÕES DAS ROTAS

Dado um grafo $G = (V, E)$ orientado e com capacidade nas arestas e um conjunto de caminhos no grafo G, não necessariamente disjuntos, definimos interseções permitidas como:

1. Caminhos que contêm vértices que podem ser acessados por uma ou mais vias distintas e a partir dos quais parte apenas uma via.

2. Caminhos que contêm vértices que são acessados por apenas uma via e a partir dos quais

parte uma ou mais vias.

Note que para cada caminho o grau de entrada e de saída de cada vértice intermediário é exatamente 1 (um). Mas a definição de interseções permitidas faz com que caminhos diferentes possam compartilhar vértices em algumas situações específicas. A Figura 5 apresenta exemplos de interseções que são permitidas e de interseções que não são permitidas.

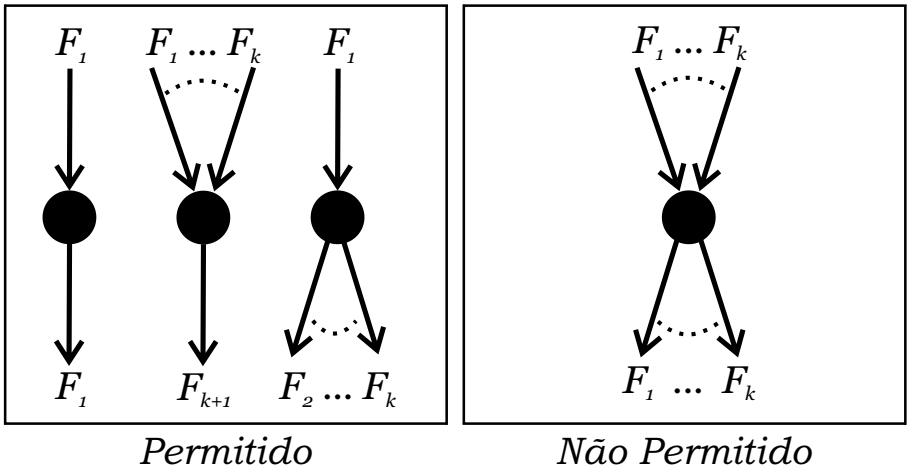

Figura 5 – Restrições nas Rotas.

Fonte: Autoria própria

Tais restrições podem causar uma solução menos eficiente do que aquelas que permitem qualquer interseção do conjunto de arestas dos caminhos, visto que algumas arestas se tornam inutilizáveis para que não haja cruzamento entre os fluxos. Entretanto o cruzamento de dois ou mais fluxos de pessoas em uma situação de pânico pode causar confusão e desordem, comprometendo assim a integridade ou a possibilidade de evacuação das mesmas. Assim essas restrições tem o intuito de evitar tais situações.

2.3 REPRESENTAÇÃO DO MAPA

O mapa do local contendo as vias, regiões onde estão as pessoas e regiões seguras, assim como as informações de capacidade de fluxo das vias e capacidade dos locais seguros será modelado como um grafo orientado $\overrightarrow{G} = (V, E)$ com capacidade nas arestas e nos vértices. As arestas do grafo representam as vias do mapa, onde o sentido da aresta representa o sentido de fluxo na via, o peso da aresta é a capacidade do fluxo da via por unidade de tempo pré-determinada. Por sua vez, os vértices do grafo representam os locais ligados pelas vias. Neste modelo, os vértices também possuem peso, este número pode ser positivo ou negativo, determinando a capacidade do local em receber mais pessoas. O peso do vértice $v \in V(G)$ é denotado por $c_v(v)$, podendo ser positivo ou negativo:

- **Positivo -** representa quantas pessoas devem ser retiradas do local no momento. Um vértice v com capacidade igual a 30 representa que ainda existem 30 pessoas que devem ser retiradas desse local. Nesse trabalho, considera-se que vértices com a capacidade positiva podem fazer parte dos caminhos de fuga desde que os mesmos não sejam os destinos finais.

- **Negativo -** representa quantas pessoas ainda é possível abrigar nesse local no momento. Um vértice v com capacidade igual à -30 representa que 30 pessoas ainda podem ser trazidas para esse local. Esses vértices podem tanto fazer parte do caminho como também ser destino final, entretanto não é obrigatório que esses locais tenham peso igual a zero ao final da execução do modelo, já que são vértices de destino na rede.

Eventualmente, para obter as rotas ideais de escoamento da população, os sentidos de algumas vias deverão ser alterados. A partir das rotas fornecidas pelo sistema, os responsáveis pela execução do plano de evacuação poderão realizar as ações necessárias para assegurar o novo sentido nas vias específicas.

2.3.1 Unidade de Transporte

Tratando-se de planejamento de rotas de fuga, é de extrema importância levar em consideração a forma como as pessoas irão fugir (a pé, de carro, de moto, de ônibus, etc.). Entretanto torna-se complexo tratar todas essas possibilidades em um método de planejamento de rotas de fuga, visto que existem diversas variáveis a serem levadas em conta:

- **Tipo de veículo -** O tipo de veículo que cada pessoa irá utilizar influencia diretamente em quantas pessoas fogem por veículo (um carro de passeio tem capacidade para 4 pessoas, entretanto um ônibus pode levar 44 pessoas).

- **Capacidade de Fluxo da via -** as medidas de fluxo em uma via são calculadas levando em consideração o meio de transporte mais popular nessa via. Em rodovias por exemplo o fluxo é medido em carros por hora, dessa forma a informação de quantos ônibus passam por hora é desconhecida.

- **Propriedade do meio de transporte -** ainda em consideração ao meio de transporte, é importante considerar se o meio de transporte é privado ou público e quais pessoas irão utilizá-lo ou é um meio de transporte público, e quais pessoas respectivamente irão utilizar cada um desses meios de transporte.

- **Estado de conservação -** o método de determinação de rotas de fuga será executado e as rotas publicadas antes de ocorrer a catástrofe. Dessa forma, diversos aspectos em relação

aos meios de transporte podem ter sofrido alterações: mudança de proprietário, veículo quebrado, veículo sem combustível, etc.

Neste trabalho o meio de transporte será considerado único e chamando de Unidade de Transporte (UT). Além disso, consideramos que as unidades de medida para o cálculo da capacidade das vias são UT por unidade de tempo. Com esta suposição, a capacidade de passageiros da UT é constante.

Dadas as restrições impostas ao modelo, os detalhes sobre a especificação da entrada e da saída do método apresentado para elaboração das rotas de fuga é apresentado na Seção 4.1.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo busca descrever o contexto em que o problema de determinar as rotas de fuga para populações em situação de catástrofe está inserido, juntamente com as teorias já existentes que podem ser utilizadas na resolução do mesmo.

3.1 PRINCIPAIS DEFINIÇÕES

Nesta seção são apresentados conceitos essenciais utilizados no desenvolvimento desse projeto e os algoritmos mais importantes considerados na abordagem do problema.

3.1.1 Grafo e Rede

Define-se um grafo G por um conjunto de vértices $V(G)$ e um conjunto de arestas $A(G)$, sendo que uma aresta é um par não ordenado $a = \{v_1, v_2\}$, v_1 e $v_2 \in V(G)$. Quando não houver ambiguidade, $V(G)$ e $A(G)$ serão denotados respectivamente por V e A. Um grafo orientado $\overrightarrow{G}$ tem suas arestas compostas por pares ordenados (v_1, v_2), onde uma aresta representa a direção de v_1 para v_2. Grafos orientados também são conhecidos por grafos direcionados ou digrafos.

A Figura 6 mostra a representação gráfica de um grafo e de um grafo orientado.

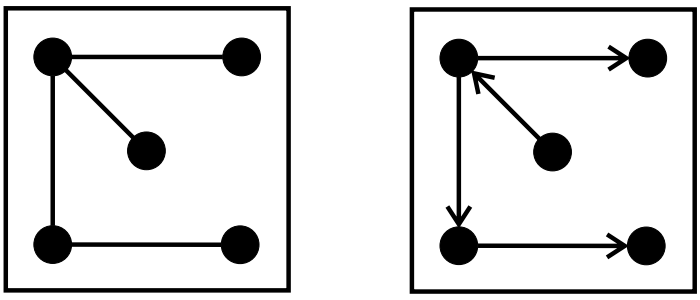

Figura 6 – Grafo e Grafo Orientado (Digrafo).

Fonte: Autoria própria

Feofiloff (2004) define uma rede R como um grafo orientado $\overrightarrow{G}$ que tem associado às suas arestas ou vértices valores numéricos. Grafos com pesos nas arestas são chamados de Grafos Ponderados. Esses valores são dados por uma função $f : a \to R, a \in A$ e representam capacidade, custo ou qualquer outro valor relevante para a aresta ou vértice. Abaixo a Figura 7 apresenta um grafo rede (ou simplesmente rede) com pesos nas arestas.

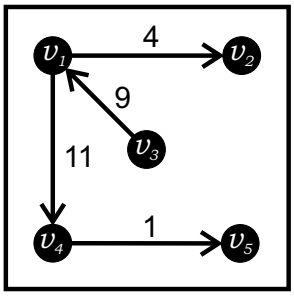

Figura 7 – Rede R - um grafo orientado e ponderado.

Fonte: Autoria própria

Redes são utilizadas para modelar problemas em que alguma informação da aresta ou do vértice deve ser levada em conta, tais como problemas de caminhos (mínimo, máximo, menor custo, etc.), escoamento de fluxo, dentre outros.

3.1.2 Caminho

Segundo Feofiloff (2004), um *caminho* p em um grafo G é apresentado como uma sequência de vértices $(v_0, v_1, ..., v_n)$ tal que $\{v_i, v_{i+1}\} \in A$, $0 \leq i < n$ e $v_i \neq v_{i+1}$, para todo $i \neq j$. A sequência (v_1, v_4, v_5) é um caminho na rede da Figura 7.

Dois caminhos p_1 e p_2 são considerados *disjuntos* se não compartilham vértices, ou seja, se não existe um vértice v_i tal que $v_i \in p_1$ e $v_i \in p_2$.

O Problema do Caminho Mínimo busca determinar qual o caminho com a menor quantidade de arestas entre dois vértice de G. Na literatura a técnica de Busca em largura é um método eficiente de resolver o problema do Caminho Mínimo, essa técnica é apresentada na próxima sessão.

3.1.2.1 Busca em largura

A Busca em Largura é um dos algoritmos elementares na Teoria de Grafos. Por meio de uma busca sistemática determina todos os vértices que são acessíveis a partir de uma origem s. Para cada vértice acessível a partir de s a Busca em Largura determina qual o caminho mínimo entre esses dois vértices.

Os vértices são descobertos segundas sua distância em relação à s, sendo descobertos primeiro os vértices com distância k, depois os com distância $k + 1$, $k + 2$, $k + i$. Um vértice nunca é visitado duas vezes, para isso assim que um vértice é descoberto pela Busca em Largura o mesmo é marcado para que não seja visitado novamente.

O algoritmo faz uso de uma fila para controle de qual vértice está buscando os vizinhos. Inicialmente o vértice s é adicionado na fila, as iterações ocorrem até que a fila esteja vazia. A cada iteração são visitados todos os vértices vizinhos do vértice que é o primeiro da fila, se o vizinho ainda não foi visitado o mesmo é incluído na fila e marcado como visitado, após visitar todos os vizinhos do primeiro vértice da fila este é removido.

O Algoritmo 1 apresenta uma implementação do algoritmo de busca em largura.

Algoritmo 1: BUSCA EM LARGURA.

```
   Entrada: G, s
 1 início
 2     insira s na fila
 3     visitado[s] ← sim
 4     distancia[s] ← 0
 5     enquanto a fila não for vazia faça
 6         v ← primeiro vértice da fila
 7         para cada vértice v_i vizinho de v faça
 8             se visitado[v_i] = não então
 9                 visitado[v_i] ← sim
10                 insira v_i na fila
11                 distancia[v_i] ← distancia[v]+1
12             fim
13         fim
14     fim
15 fim
```

Note que no algoritmo o caminho entre s e todos os vértices alcançáveis a partir dele não é salvo, entretanto é possível mediante a inclusão de um vetor guardar a partir de qual vértice cada vértice é acessado, guardando assim os caminhos mínimos.

3.1.3 Fluxo

Como pode ser visto em Cormen *et al.* (2002), dada uma rede $\overrightarrow{G} = (V, E)$ em que cada aresta $(u, v) \in E$ tem uma capacidade $c(u, v) \geq 0$, e dois vértices, uma origem s e um destino (ou sorvedouro) t, um fluxo em G é uma função de valor real $f : V \times V \rightarrow R$ que satisfaz as propriedades:

- **Restrição de capacidade -** Para todo $u, v \in V$, $f(u, v) \leq c(u, v)$.
- **Conservação de fluxo -** Para todo $u \in V \setminus \{s, t\}, \sum_{v \in V} f(u, v) = \sum_{v \in V} f(v, u)$.

O valor da função $f(u, v)$ é chamado de fluxo do vértice u até o vértice v. O valor de um fluxo f_R de uma rede R é definido como $|f_R| = \sum_{v \in V} f(s, v)$. Note que $|f_R| = \sum_{v \in V} f(v, t)$, onde s é o vértice de origem e t é o vértice de destino.

A restrição de capacidade garante que o fluxo presente em uma aresta não exceda a capacidade da aresta. A conservação de fluxo garante que o fluxo que chega em um vértice é o mesmo fluxo que sai desse vértice, exceto nos vértices de origem e destino.

O fluxo total que entra em um vértice v é definido por:

$$\sum_{u \in V} f(u, v).$$

O fluxo total que sai de um vértice v é definido como:

$$\sum_{u \in V} f(v, u).$$

No Problema de Fluxo Máximo busca-se determinar qual o maior fluxo que é possível escoar em uma rede dada uma origem s e um destino t. Note que o valor do fluxo máximo é obtido através do valor máximo que a função $\sum_{v \in V} f(s, v)$ pode assumir.

3.1.3.1 Exemplo: Problema de Fluxo Máximo

Um dos problemas clássicos de fluxo é o problema de transportes. Nesta seção será apresentado um problema de transporte baseado em Cormen *et al.* (2002). Dada uma empresa que tem sua fábrica na cidade de Macedônia e seu estoque na cidade de Jales, essa empresa aluga espaço em caminhões de transportadoras para levar seus produtos fabricados para o estoque. Cada caminhão tem sua rota definida e só pode dispor de certo espaço para a empresa transportar seus produtos, dessa forma o número de produtos que a empresa pode transportar por dia em cada rota é limitado. Para evitar produzir um número maior do que o que pode ser transportado até o estoque, a empresa deseja saber qual a quantidade máxima de produto que é possível transportar da fábrica até o depósito sem se importar com as rotas.

Pode-se modelar esse problema como um problema de fluxo em rede, a Figura 8 trás a representação da rede que modela o problema.

A partir dessa rede, para dar a resposta à empresa basta encontrar o Fluxo Máximo do vértice s até o vértice t. Na Figura 9 apresenta-se o fluxo máximo que soluciona o problema.

Note que esse fluxo atende as propriedades da definição de fluxo mencionadas na seção 3.1.3:

- **Restrição de capacidade -** Em cada aresta é alocada uma quantidade de fluxo menor ou igual à sua capacidade. Tomando como exemplo a aresta (v_3, v_t) que tem capacidade 20, na solução essa aresta transporta um fluxo de valor 15, que respeita a sua capacidade.
- **Conservação de fluxo -** Em cada vértice da rede, exceto no vértice inicial e no vértice final, é possível observar que a soma dos fluxos que chegam ao vértice é igual à soma dos fluxos que saem do vértice. O fluxo em v_3 é:

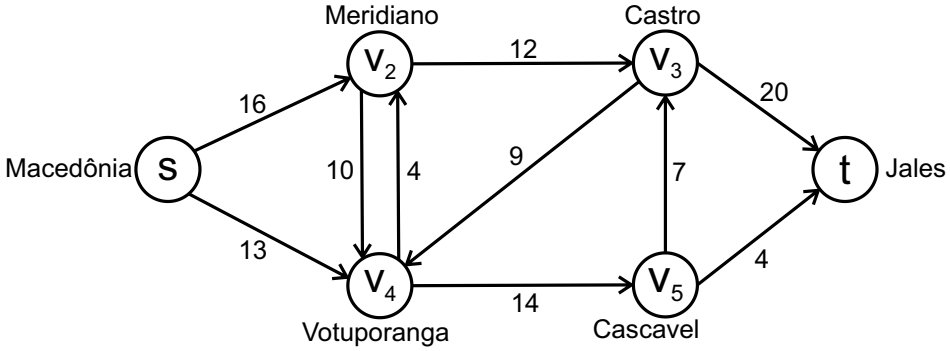

Figura 8 – Exemplo de rede para o Problema de Transporte.

Fonte: Autoria própria

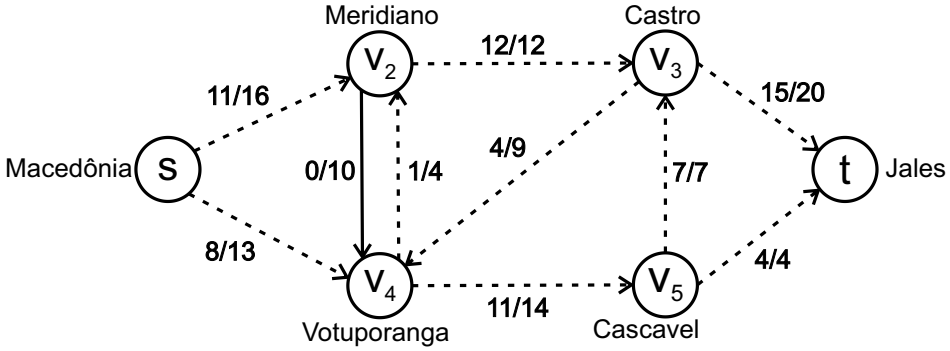

Figura 9 – Resposta para o exemplo de Problema de Transporte da Figura 8.

Fonte: Autoria própria

$$\sum_{u \in V} f(v_3, u) = \sum_{u \in V} f(u, v_3) =$$
$$(f(v_3, v_4) + f(v_3, t)) = (f(v_2, v_3) + f(v_5, v_3)) =$$
$$(4 + 15) = (12 + 7) = 19$$

Na seção 3.2 será apresentado um algoritmo capaz de solucionar o problema do Fluxo Máximo.

3.1.3.2 Fluxo com várias origens e vários destinos

Ao trabalhar com fluxo, pode ser necessário escoar um fluxo de mais de uma origem para mais de um destino. Felizmente esse caso não é mais difícil que o caso onde temos apenas uma origem e um destino.

Uma maneira bastante conhecida de resolver o problema do fluxo máximo com múltiplas origens e/ou múltiplos destinos é, dada uma rede com k vértices de origem $s_1, s_2, \ldots, s_k$ e

q vértices de destino $t_1, t_2, \ldots, t_q$, cria-se uma superorigem S e um superdestino T. Para cada origem s_i, $1 \leq i \leq k$, cria-se uma aresta orientada com capacidade infinita da superorigem S até o vértice inicial s_i. Para cada vértice de destino t_i, $1 \leq i \leq q$, cria-se uma aresta orientada com capacidade infinita do vértice final t_i até o superdestino T. A Figura 10 apresenta a rede com os vértices superorigem e superdestino.

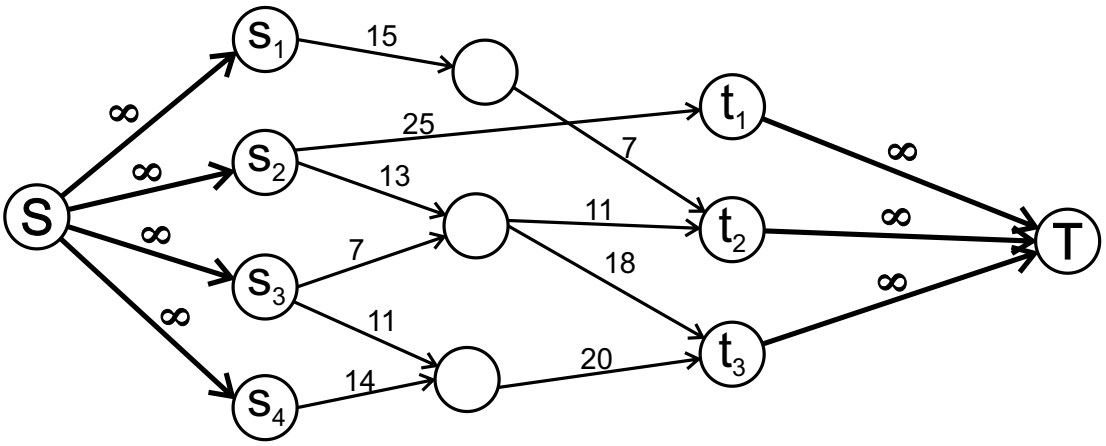

Figura 10 – Fluxo com várias origens e vários destinos.

Fonte: Autoria própria

Como as capacidades das novas arestas são infinitas, a superorigem fornece todo fluxo que as origens possam vir a consumir e o superdestino consome todo o fluxo que os destinos possam vir a produzir.

3.2 MÉTODO DE FORD-FULKERSON

O método de Ford-Fulkerson é capaz de encontrar o fluxo máximo presente em uma rede G. É chamado de método e não de algoritmo por ser uma ideia geral que pode ser implementada de diferentes maneiras e com diferentes complexidades.

O método é iterativo, sendo executado enquanto houver uma caminho capaz de aumentar o fluxo entre a origem s e o destino t. Inicialmente o fluxo é considerado 0, ou seja $f(u, v) = 0, \forall\ u, v \in V$. A cada iteração tenta-se aumentar o somatório dos fluxos que saem de s. O método tem seu fim quando não existe um novo caminho capaz de aumentar o fluxo da rede de s até t para ser explorado, chegando assim no fluxo máximo de G.

As seções 3.2.1, 3.2.2 e 3.2.3 trazem os conceitos nos quais o método de Ford-Fulkerson se baseia. Na seção 3.2.4 é apresentado um algoritmo que implementa o método de Ford-Fulkerson.

3.2.1 Redes residuais

A *capacidade residual* de uma aresta é definida pela quantidade de fluxo que ainda é possível passar por ela antes de atingir a capacidade da aresta. Representada por c_f, a capacidade residual da aresta (u, v), $u, v \in V$ é definida por:

$$c_f = c(u, v) - f(u, v).$$

Considere uma aresta (u, v) em que $c(u, v) = 10$ e que $f(u, v) = 11$, ou seja, a capacidade da aresta é de 10 e já existe um fluxo de valor 11 passando por essa aresta, então a capacidade residual da aresta é de $c_f(u, v) = 9$.

Nos casos em que a rede possui duas arestas (u, v) e (v, u), então $c_f(u, v) = c(u, v) - f(u, v) + f(v, u)$. Ainda considerando o exemplo anterior, se existe um $f(v, u) = 8$, então $c_f(u, v) = 20 - 11 + 8 = 17$. Note que, se $f(u, v) = 0$, então $c_f(u, v) = 20 - 0 + 8 = 28$, ou seja, a capacidade residual, pode em alguns casos, ser maior que a própria capacidade da aresta.

Uma *rede residual* $G_f(V, A_f)$, resultante da ação de um fluxo f sobre uma rede G, é composta pelo conjunto de vértices do grafo original $V(G)$ e pelo conjunto de arestas $A_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$, onde a capacidade $c(u, v)\ \forall\ u, v \in V(G_f)$ é dada pela capacidade residual da aresta no grafo original.

Se $(u, v) \notin A$ e $(v, u) \notin A$, então $c(u, v) = c(v, u) = 0$, $f(u, v) = f(v, u) = 0$ e $c_f(u, v) = c_f(v, u) = 0$. Dessa forma, dados dois vértices $u, v \in V$, se não existe aresta que os ligue em G, então também não existe aresta que os ligue em G_f. Portanto, $|A_f| \leq 2|A(G)|$.

A Figura 11 apresenta a rede residual G_f obtida a partir da rede G e do fluxo f.

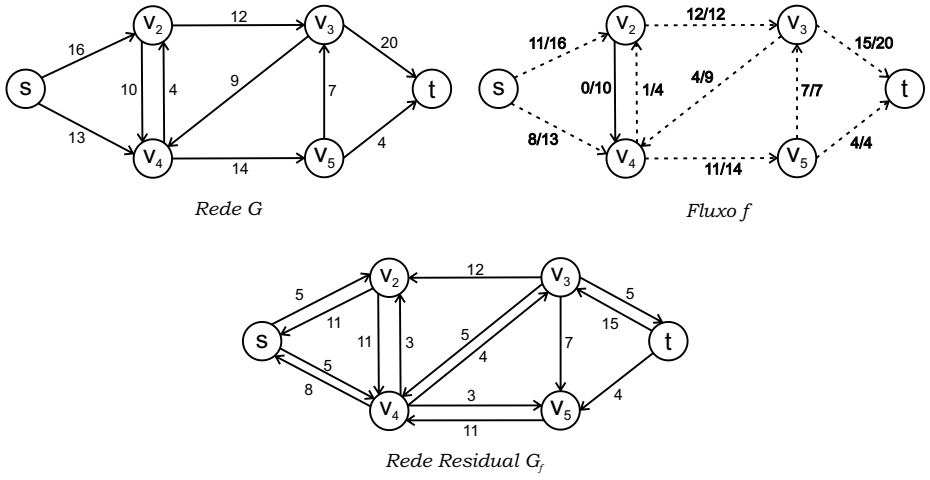

Figura 11 – Rede Residual G_f.

Fonte: Autoria própria

A seguir o Lema 1 relaciona o fluxo no grafo G com a rede residual G_f. Tal lema foi

provado por Cormen *et al.* (2002).

Lema 1. *[Cormen et al. (2002)] Seja $G = (V, A)$ uma rede com origem s e destino t, e seja f um fluxo em G. Seja G_f a rede residual de G, resultante da ação de f, e seja f' um fluxo em G_f. Então, a soma de fluxo $f + f'$ definida pela equação $(f + f')(u, v) = f(u, v) + f'(u, v)$ é um fluxo em G com valor $|f| + |f'|$.*

Observe que como $f + f' > f$, então $f + f'$ esta mais próximo do fluxo máximo que f, visto que f' é o fluxo da rede residual e que as capacidades das arestas na rede residual representam quanto fluxo ainda é possível passar por cada aresta, levando em conta a rede G e o fluxo f.

3.2.2 Caminhos aumentantes

Dada uma rede $G = (A, V)$, um fluxo f e uma rede residual G_f resultante da ação de f em G, um *caminho aumentante* p é um caminho com origem s e destino t na rede residual. Pela definição de rede residual, a capacidade das arestas em uma rede residual é a quantidade de fluxo que ainda é possível passar por cada aresta. Assim cada aresta (u, v) do caminho aumentante permite a passagem de fluxo adicional na aresta (u, v) da rede G em conformidade com a propriedade de capacidade da aresta.

A quantidade máxima pela qual é possível aumentar o fluxo em cada aresta de um caminho aumentante p é chamada de *capacidade residual* de p, é denotada por $c_f(p)$ e é dada por:

$$c_f(p) = min\{c_f(u, v) : (u, v) \text{ está em } p\}.$$

A Figura 12 apresenta uma rede residual G_f obtida através da ação do fluxo f na rede G. Na rede residual G_f o caminho em destaque é um caminho aumentante.

No exemplo da Figura 12 é possível passar mais 4 unidades de fluxo em cada aresta do caminho aumentante, visto que a menor capacidade residual é $c_f(v_4, v_3) = 4$.

Pela definição de caminho aumentante, esse fluxo a mais pode ser transportado na rede G pelas mesmas arestas do caminho aumentante, assim o fluxo f_2 na Figura 12 apresenta o fluxo acrescido das 4 unidades. O fluxo da aresta (v_3, v_4) caiu de 4 para 0 pois o fluxo que sofreu acréscimo de 4 unidades foi o fluxo no sentido oposto, de v_4 para v_3.

Cormen *et al.* (2002) apresenta o Corolário 1, que mostra que quando adicionamos o fluxo f_p ao fluxo f obtemos um fluxo em G mais próximo do fluxo máximo da rede G.

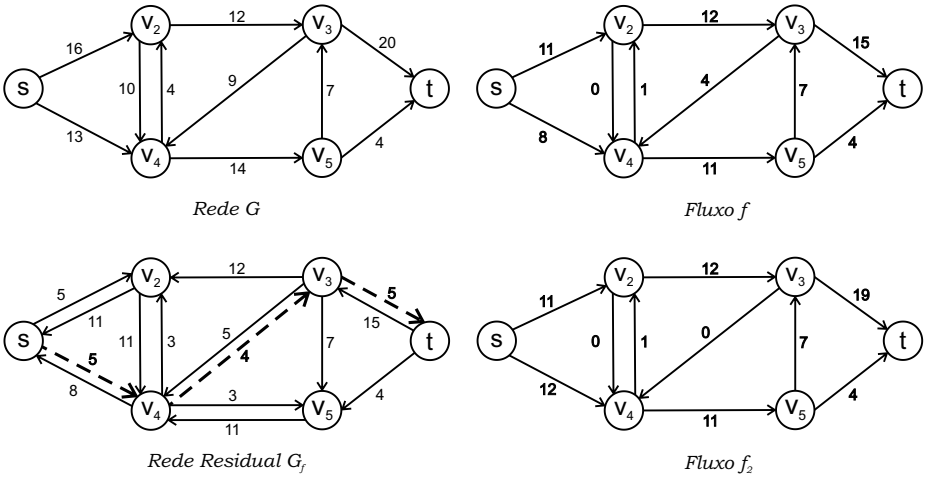

Figura 12 – Caminho Aumentante em G_f.

Fonte: Autoria própria

Corolário 1 (Cormen)**.** *Seja $G = (V, A)$ uma rede, seja f um fluxo em G e seja p um caminho aumentante em G_f. Seja f_p definido como*

$$f_p(u,v) = \begin{cases} c_f & \textit{se} \quad (u,v) \textit{ está em } p, \\ -c_f(p) & \textit{se} \quad (v,u) \textit{ está em } p, \\ 0 & \textit{em} \quad \textit{caso contrário.} \end{cases}$$

Defina uma função $f' : V \times V \to R$ por $f' = f + f_p$. Então, f' é um fluxo em G com valor $|f'| = |f| + |f_p| > |f|$.

Pelo Corolário 1, o valor da soma do fluxo f em G com o fluxo f_p é maior que o fluxo f. Como o valor de f aumenta, ele se torna mais próximo do valor do fluxo máximo.

3.2.3 Corte de fluxo em redes

Um *corte (S,T)* em uma rede $G = (V, A)$ particiona o conjunto dos vértices V em dois conjuntos S e T de forma que o vértice origem s pertença a S, e o vértice destino t pertença a T e os subgrafos induzidos $G[S]$ e $G[T]$ sejam conexos. Seja f um fluxo em G, o *fluxo líquido* pelo corte (S, T) é definido pela soma dos fluxos das arestas que saem de S em direção a T, ou seja:

$$f(S,T) = \sum_{u \in S,\ v \in T} f(u,v).$$

A *capacidade* do corte $c(S, T)$ é definida pela soma das capacidades das arestas que saem de S em direção à T, ou seja:

$$c(S,T) = \sum_{u \in S,\ v \in T} c(u,v)$$

Um *corte mínimo* em uma rede é um corte (S^*, T^*) tal que, $c(S^*, T^*) = min\{c(S,T), \forall\ (S,T) \in G\}$.

A Figura 13 apresenta um exemplo de corte (S,T) na rede G. No exemplo os vértices do conjunto S são $\{s, v_4\}$. Por sua vez os vértices em T são $\{v_2, v_3, t\}$. As arestas contidas no corte (S,T), ou seja, as arestas que saem de S em direção à T são $(s,v_2), (s,v_3), (v_4,v_3)$.

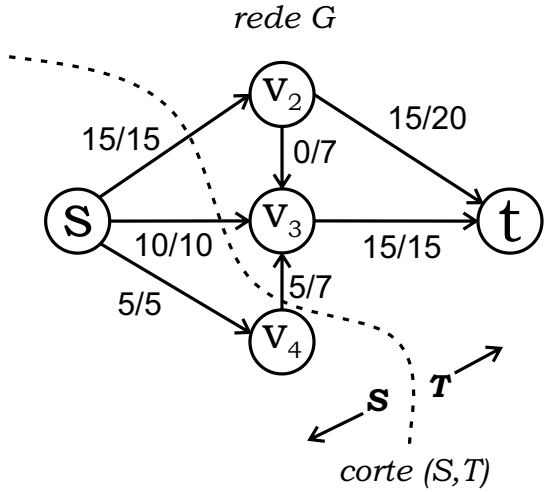

Figura 13 – Corte (S,T) na rede G.

Fonte: Autoria própria

No exemplo da Figura 13 temos que o valor do fluxo líquido $f(S,T)$ é

$$\begin{aligned} f(S,T) &= \sum_{u \in S,\ v \in T} f(u,v) \\ &= f(s,v_2) + f(s,v_3) + f(v_4,v_3) \\ &= 15 + 10 + 5 \\ &= 30. \end{aligned}$$

A capacidade do corte (S,T) do exemplo é

$$\begin{aligned} c(S,T) &= \sum_{u \in S,\ v \in T} c(u,v) \\ &= c(s,v_2) + c(s,v_3) + c(v_4,v_3) \\ &= 15 + 10 + 7 \\ &= 32. \end{aligned} \tag{3.1}$$

Perceba que o corte (S, T) apresentado no exemplo da Figura 13 não é o corte mínimo da rede, visto que o corte $(S', T') = \{(s, v_2), (s, v_3), (s, v_4)\}$ tem capacidade $c(S', T') = 30$, sendo este o corte mínimo.

A seguir, é apresentado o famoso Teorema do Fluxo Máximo e Corte Mínimo, segundo o qual o valor do fluxo máximo em uma rede é igual ao corte mínimo. A demonstração apresentada pode ser encontrada em Camponogara (2005).

Teorema 1 (Teorema do fluxo máximo e corte mínimo)**.** *Dada uma rede G, um fluxo f em G e a rede residual G_f. O valor do fluxo máximo que é possível transportar de um vértice $s \in V(G)$ para um vértice $t \in V(G)$ é igual a capacidade $c(S, T)$ do corte mínimo em G.*

Demonstração. Seja $f(s, t) = k$ o fluxo máximo em G. Seja o conjunto U composto por todos os vértices de G_f possíveis de serem alcançados a partir de s e $\overline{U} = V - S$. Como em G_f não existe um caminho aumentante (porque o fluxo f é máximo) o valor da capacidade $c(U, \overline{U})$ é k. Uma vez que $k \leq min\{c(S, T) : (S, T)$ é um corte em $G\}$, conclui-se que $k = min\{c(S, T) : (S, T)$ é um corte em G. □

3.2.4 Algoritmo de Ford-Fulkerson

A partir do método de Ford-Fulkerson, pode-se construir o Algoritmo 2, apresentado a seguir. Chamaremos tal algoritmo de *Algoritmo de Ford-Fulkerson*.

Observe que o algoritmo de Ford-Fulkerson executa uma série de iterações em uma rede G, em cada iteração é descoberto um novo caminho aumentante p na rede residual G_f e o fluxo f em G é aumentado pela capacidade residual do caminho $c_f(p)$. Quando não há mais caminhos aumentantes a serem explorados o algoritmo se encerra.

O algoritmo faz uso do Corolário 1 para obter o fluxo máximo em uma rede G. O Algoritmo 2 apresenta um pseudocódigo do algoritmo de Ford-Fulkerson.

No Algoritmo de Ford-Fulkerson o laço da linha 2 até a linha 5 garante a inicialização do fluxo com valor 0. O segundo laço, da linha 6 até a 12 trata de buscar caminhos aumentantes na rede residual G_f. Uma vez encontrado tal caminho, a instrução da linha 7 determina a capacidade residual deste caminho como sendo a menor capacidade entre as arestas contidas no caminho aumentante. Determinada a capacidade residual, o terceiro laço da linha 8 a 11 atualiza o valor do fluxo em cada aresta de G que está contida no caminho residual. Na linha 10 o fluxo de v para u recebe o valor negativo do fluxo de u para v.

Algoritmo 2: FORD-FULKERSON (CORMEN *et al.*, 2002).

```
   Entrada: G, s, t
1  início
2      para cada aresta (u, v) ∈ A(G) faça
3          f[u][v] ⟵ 0
4          f[v][u] ⟵ 0
5      fim
6      enquanto existir um caminho aumentante p em G_f faça
7          c_f(p) = min {c_f(u, v) : (u, v) ∈ p}
8          para cada aresta (u, v) ∈ p faça
9              f[u][v] ⟵ f[u][v] + c_f(p)
10             f[v][u] ⟵ −f[u][v]
11         fim
12     fim
13 fim
```

3.2.4.1 Exemplo de Ford-Fulkerson

A Figura 14 apresenta uma rede G com vértice inicial s e vértice destino t. Nesta figura é possível acompanhar a execução do Algoritmo de Ford-Fulkerson para a determinação do fluxo máximo que pode-se escoar de s até t respeitando as capacidades das arestas de G.

Inicialmente os valores dos fluxos das arestas da rede G são 0 e são executados os passos para criar a rede residual G_f.

A primeira iteração encontra o caminho aumentante $p = ((s, v_2), (v_2, t))$, destacado em tracejado na Figura 14. A capacidade aumentante desse caminho é $c_f(p) = min\{c_f(s, v_2), c_f(v_2, t)\} = min\{15, 20\} = 15$. Assim, o fluxo f é atualizado em cada aresta de p, então $f(s, v_2) = f(s, v_2) + c_f(p) = 0 + 15 = 15$ e $f(v_2, t) = f(v_2, t) + c_f(p) = 0 + 15 = 15$.

Na segunda iteração é descoberto o caminho aumentante $p = ((s, v_4), (v_4, v_3), (v_3, t))$, tal caminho tem capacidade aumentante $c_f(p) = 5$, assim os fluxos das arestas contidas em p são atualizados para $f(s, v_4) = 5$, $f(v_4, v_3) = 5$ e $f(v_3, t) = 5$.

A terceira iteração encontra o caminho aumentante $p = ((s, v_3), (v_3, t))$, tal caminho tem capacidade aumentante $c_f(p) = 10$, assim os fluxos das arestas contidas em p são atualizados para $f(s, v_3) = 10$ e $f(v_3, t) = f(v_3, t) + c_f(p) = 5 + 10 = 15$.

Na quarta iteração não existem mais caminhos aumentantes a serem explorados. Dessa forma o algoritmo se encerra tendo encontrado o fluxo máximo f em G.

Pelas definições apresentadas na seção 3.1.3 o fluxo total em uma rede é a soma de todos os fluxos que saem da origem. Dessa forma o fluxo máximo encontrado pelo algoritmo pode ser calculado da seguinte forma.

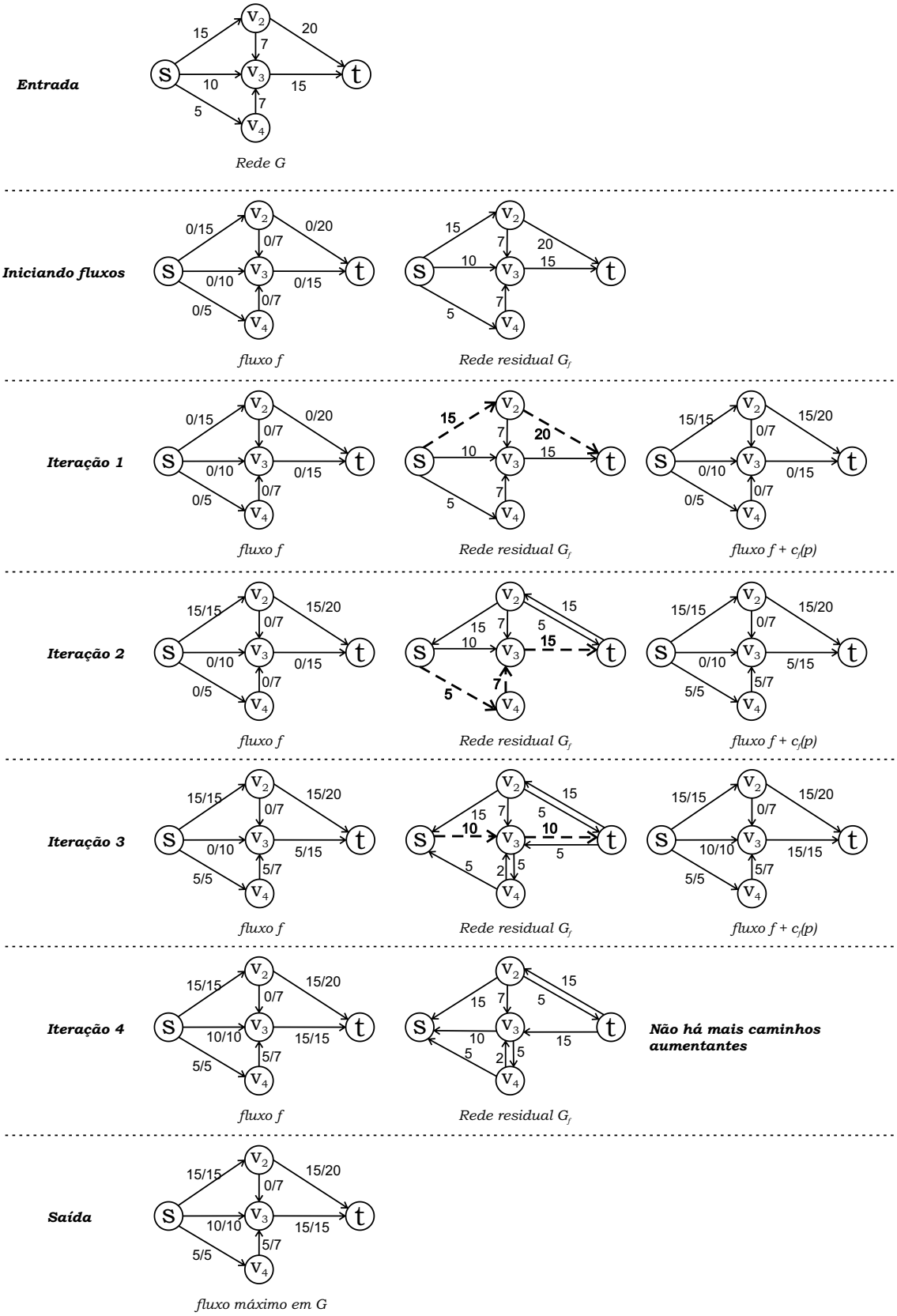

Figura 14 – Exemplo execução algoritmo de Ford-Fulkerson.

Fonte: Autoria própria

$$
\begin{aligned}
|f_R| = \sum_{u \in V} f(s,u) &= f(s,v_2) + f(s,v_3) + f(s,v_4) \\
&= 15 + 10 + 5 \\
&= 30
\end{aligned}
$$

A seção 3.2.5 mostra que o algoritmo de Ford-Fulkerson tem um teto superior de $O(|A|f)$ iterações, onde f é o fluxo máximo encontrado pelo algoritmo. Nesse exemplo foram necessárias 4 iterações, que é um número abaixo do limite de pior caso. Considerando o exemplo dado, temos $|A|f = 7 \times 30 = 210$.

3.2.5 Análise de Complexidade do Método Ford-Fulkerson

O fator crucial para determinar a complexidade do Método de Ford-Fulkerson é o método utilizado para descobrir novos caminhos aumentantes. Se este método for mal implementado o algoritmo pode nem sequer ser finalizado. Partindo do pressuposto de que o caminho aumentante é escolhido de forma arbitrária temos a complexidade de pior caso $O(|A|f)$, onde f é o fluxo máximo encontrado pelo algoritmo.

O primeiro laço do Algoritmo 2 (linhas 2 a 5) é executado para todas as arestas da rede G, portanto é executado $\Theta(|A|)$ vezes.

O segundo laço (linhas 6 a 12) é executado enquanto existir um caminho aumentante, ou seja, enquanto o fluxo f em G ainda pode ser aumentado. Note que no pior caso o fluxo f vai ser aumentado de uma em uma unidade, portanto esse laço é executado $O(f)$ vezes.

O terceiro laço é executado para cada aresta presente no caminho aumentante, no pior caso, todas as arestas de G estão presentes no caminho aumentante, portanto esse laço é executado $O(|A|)$ vezes.

Como o terceiro laço está dentro do segundo, a complexidade de pior caso do algoritmo é $O(f + |A|f)$. Logo a complexidade do algoritmo é $O(|A|f)$. Ou seja, independentemente da forma com que a descoberta de caminhos aumentantes for implementada, o método terá no máximo $|A| \times f$ iterações.

Visando melhorar a descoberta de novos caminhos aumentantes, diversas implementações para Ford-Fulkerson foram propostas. Em Feofiloff (2004) é possível encontrar uma análise detalhada de todos os algoritmos aqui mencionados.

O algoritmo de *Dinits* foi proposto em 1970 (DINITS, 1970) como uma implementação do Método de Ford-Fulkerson. Ele implementa a descoberta de caminhos aumentantes com uma busca em largura, entretanto os resultados das buscas não são descartados, dessa forma uma nova busca em largura começa onde a anterior parou. Com essa implementação o algoritmo obtêm uma complexidade $O(|V|^2|A|)$.

Implementando uma busca em largura (mas descartando os resultados anteriores) como método para descoberta de novos caminhos aumentantes em 1972 foi proposto o algoritmo *Edmonds-Karp* (EDMONDS; KARP, 1972), algoritmo este com complexidade $O(|V||A|^2)$.

Existem ainda algoritmos diferentes, que não são baseados em Ford-Fulkerson (fluxo incrementado pela capacidade residual do caminho aumentante). O algoritmo *FIFO Preflow-push* proposto em Goldberg (1985), implementa o conceito de *pré-fluxo* através de uma fila, obtendo uma complexidade de $O(|V|^3)$.

3.3 TRABALHOS RELACIONADOS

Antes de desenvolver um método novo para o problema, faz-se necessário verificar na literatura quais os métodos já existentes. Dessa forma é possível verificar a necessidade do desenvolvimento de um novo método.

Na literatura existem métodos de determinação de rotas para uma população em área de risco, como é possível ver em Hutchinson (1979), Martin e Manheim (1965) e Papacostas e Prevedouros (1993). Nesse trabalho é descrito o método criado por Campos em 1997, visto que este método apresenta mais similaridade com o método proposto, pois também modela o problema como um problema de escoamento de fluxo.

Existem outras maneiras de modelar o problema de determinar rotas de fuga para uma população em área de risco. *Algoritmos Genéticos* são amplamente conhecidos e muito utilizados em diferentes contextos. Em Saadatseresht, Mansourian e Taleai (2009) os autores utilizam uma implementação do algoritmo genético *NSGA-II* (DEB *et al.*, 2002) para determinar o escoamento de fluxo com múltiplas origens e/ou destinos que define as rotas de fuga da população.

Outra possibilidade é a modelagem do problema utilizando *Sistemas Multiagentes* através de simulações computacionais. Pesquisadores da PUCRS (Pontifica Universidade Católica do Rio Grande do Sul) desenvolveram a ferramenta *CrowdSim* que simula computacionalmente o comportamento de multidões em diferentes ambientes. Em Cassol *et al.* (2012) a ferramenta é apresentada, juntamente com um estudo de caso onde a mesma foi testada para a evacuação do Estádio Municipal João Havelange, Rio de Janeiro. No estudo de caso a ferramenta constatou que a evacuação total do estádio leva em torno de 7 minutos. Entretanto a ferramenta não determina quais as rotas que a multidão deve seguir, ela apenas pode ser utilizada para testar as rotas já determinadas.

3.3.1 Determinação de rotas disjuntas para o escoamento de populações

Campos (1997) modela o problema de determinar as rotas de fuga para populações em áreas de risco como um problema de fluxo em rede, onde cada vértice representa um local e cada aresta representa uma via. Cada aresta tem dois valores a ela associados, a capacidade da aresta $c(u, v)$, que representa a quantidade de veículos que podem passar por vez pela aresta, e o tempo t, que representa o tempo mínimo para um veículo percorrer a aresta. A capacidade C_p de um caminho p é definida como a menor capacidade entre as capacidades das arestas presentes no caminho p. O tempo T_p do caminho p é definido como a soma dos tempos de todas as arestas em p.

É possível estabelecer uma relação entre a capacidade e o tempo do caminho, assim, define-se $I_p = \dfrac{C_p}{T_p}$ como o indicador de performance do caminho.

O problema tratado por Campos é encontrar k caminhos disjuntos, de forma a maximizar a relação $\sum_{p=1}^{k} I_p$, ou seja,

$$\max \left\{ \sum_{p=1}^{k} \frac{C_p}{T_p} \right\}.$$

A autora apresenta um algoritmo heurístico para a resolução do problema como contribuição original da sua tese, na seção 3.3.1.1 apresentamos um pseudocódigo que implementa o método proposto por Campos.

3.3.1.1 Algoritmo de Campos

Seja p um caminho na rede G, o *conjunto de gargalos* é definido como o conjunto de arestas em p tal que suas capacidades sejam iguais a menor capacidade de p. A identificação dos gargalos do caminho possibilita encontrar e substituir as arestas com menor capacidade. O conjunto de vértices coincidentes é definido como o conjunto de vértices que aparecem o maior número de vezes nos caminhos já encontrados pelo algoritmo. Define-se x como sendo o número de vezes que os vértices coincidentes aparecem nos caminhos.

O Algoritmo 3 apresenta o pseudocódigo do método apresentado em (CAMPOS, 1997).

O algoritmo recebe como entrada uma rede G, o vértice inicial s e o vértice destino t, sua saída é o conjunto de k-rotas disjuntas de s até t. Este algoritmo faz uso de três redes, G, G_a e G_v: na rede G_a são retiradas apenas as arestas gargalos e na rede G_v são retirados apenas os vértices coincidentes (vértices presentes em mais de um caminho). Dessa forma o algoritmo vai buscando caminhos com t mínimo em G_a e em G_v e a cada iteração arestas gargalos ou vértices coincidentes são retirados até que não haja mais caminho de s até t em G_a ou G_v. Nesse ponto

Algoritmo 3: CAMPOS.

Entrada: G, s, t
início
  $G_t \leftarrow G_a \leftarrow G_v \leftarrow G$
  $k \leftarrow$ número de caminhos disjuntos de s até t
  $n \leftarrow 2$
  $C \leftarrow \varnothing$
  enquanto *(*$n > 0$*)* **faça**
    $C_i \leftarrow$ k-Caminhos com t mínimo de s a t em G_t
    $C \leftarrow C \cup C_i$
    se $|C_i| = 0$ **então**
      $n = n - 1$
      se *iteração passada retirou gargalos* **então**
        retirar nos coincidentes
        $G_t \leftarrow G_v$
      senão
        retirar gargalos
        $G_t = G_a$
      fim
    senão
      se *iteração passada retirou gargalos* **então**
        $V_i \leftarrow$ vértices que aparecem o maior número de vezes em C
        se $x > (|C| - k - 1)$ **então**
          $G_v \leftarrow G_v - V_i$
          $n = 2$
        senão
          $n = n - 1$
        fim
        $G_t \leftarrow G_v$
      senão
        $A_i \leftarrow$ arestas que são gargalos em C_i
        $G_a \leftarrow G_a - A_i$
        $G_t \leftarrow G_a$
        $n = 2$
      fim
    fim
  fim
  $R_p \leftarrow$ todos os conjuntos de caminhos R_n | $\{|R_n| = k\}, \{c \in R_n$ e $c \in C\}$ e $\{c_1 \in R_n$ e $c_2 \in R_n$ não compartilhem vértice$\}$
  $R_f \leftarrow$ conjunto $R_n \in R_p$ com maior soma $\sum_{c \in R_n} I_p(c)$
fim
Saída: R_f

é selecionada a melhor combinação de k caminhos encontrados.

Inicialmente o algoritmo calcula a quantidade de caminhos disjuntos existentes entre s e t. Uma aproximação possível é o menor número entre a quantidade de arestas que sai de s

e quantidade de arestas que chegam em t. No seu trabalho Campos propôs um método baseado no conceito de redes em níveis para determinar quantos caminhos disjuntos existem entre s e t.

O algoritmo é executado enquanto existir um caminho de s até t em G_a ou em G_v, a cada iteração são encontrados os k-caminhos com T_p mínimo e ou são retiradas as arestas gargalos desses caminhos ou os vértices coincidentes.

A condição da linha 21 verifica se com a retirada do conjunto de vértices coincidentes ainda é possível encontrar $k-$caminhos disjuntos, visto que a retirada desses vértices ocasiona uma diminuição no número de caminhos existentes.

No final, a melhor combinação de k-caminhos é selecionada como resposta.

3.3.1.2 Aplicação Algoritmo Campos

Em seu trabalho Campos realizou uma série de testes com seu algoritmo visando avaliar o desempenho do mesmo. Em cada teste verificou-se o número de iterações necessárias fazendo-se k variar, retirando e colocando arestas e variando a direção de alguns arcos. O Quadro 1 apresenta os resultados obtidos por Campos.

Quadro 1 – Resultados da execução do Algoritmo de Campos.

Fonte: Campos (1997)

Rede	num. de caminhos	k	num. nós	num. arestas	num. iterações
BG	5092	3	46	77	11
BG1	2210	2	45	70	8
BG2	4970	2	45	73	9
BG3	5978	2	45	74	9
FCON	184	2	20	41	5
FCON2	294	3	20	43	8
FCON3 (*)	976	2	20	31	4
FCON4 (*)	3349	3	20	35	6
FCON5	387	3	20	39	5
FCON6	387	3	20	41	5
FCON7	1370	4	25	55	8 (**)

(*) redes não orientadas.
(**) encontrados apenas 3 caminhos.

Com base no quadro conclui-se que o número de iterações depende diretamente do número de caminhos disjuntos (k). Também observou-se que para $k > 2$ é possível que o algoritmo não encontre k caminhos, entretanto nesses casos o algoritmo encontra no mínimo dois caminhos.

3.3.1.3 Considerações sobre o algoritmo Campos

O algoritmo proposto por Campos é uma opção válida quanto à definição de rotas disjuntas para evacuação de uma população em área de risco. O algoritmo faz uso de rotas disjuntas para minimizar os conflitos e acidentes que podem ocorrer no cruzamento de fluxos de pessoas.

Entretanto apesar de afirmar que o algoritmo fornece as rotas ótimas, nos testes foi verificado que nem sempre o algoritmo retorna k rotas. Além disso os testes foram realizados apenas com grafos com no máximo 46 vértices, dessa forma são necessários testes com grafos maiores, que representem cidades ou estados.

Na definição do problema Campos (1997) garante que seu algoritmo apresenta as k-rotas de fuga disjuntas ótimas do conjunto de áreas de risco até o conjunto de locais seguros, entretanto o algoritmo apresenta a resposta para apenas uma área de risco e um local seguro. É possível contornar esse problema com a técnica apresentada na Sessão 3.1.3.2.

A principal diferença entre o método desenvolvido por Campos e o método proposto por esse trabalho é o suporte nativo para mais de uma área de risco e mais de um local seguro, além das restrições das rotas, nesse trabalho será proposto um relaxamento na definição de rotas disjuntas (Sessão 2.2) visando um melhor aproveitamento das arestas da rede.

4 MÉTODO DE ALOCAÇÃO DE FLUXO EM REDES PARA DETERMINAÇÃO DE ROTAS DE FUGA PARA POPULAÇÕES

O problema de determinar as rotas de fuga para uma população em áreas de risco foi modelado como um problema de escoamento de fluxo em redes. A partir da definição do modelo feita no Capítulo 2 e de toda fundamentação teórica apresentada no Capítulo 3 foi desenvolvido um método capaz de determinar as rotas de fuga para uma população em situação de catástrofe, tal método faz uso do método de Ford-Fulkerson apresentado na Sessão 3.2 para calcular o valor do fluxo máximo da rede.

Este capítulo apresenta a descrição de todas as etapas que compõe o método proposto, desde a descrição da entrada recebida na Sessão 4.1, a determinação das rotas de fuga na Sessão 4.2, o escoamento do fluxo calculado na Sessão 4.3 e o modo como o tempo de evacuação é calculado na Sessão 4.4. Ainda, na Sessão 4.5 é apresentado um pseudocódigo que implementa o método e na Sesssão 4.6 um exemplo de utilização passo a passo do método. Na Sessão 4.7 é descrito o programa em linguagem C que implementa o método.

4.1 ENTRADA DO MÉTODO

O método receberá como entrada o grafo G do local, tal grafo é não orientado, juntamente com as capacidades de cada via, o conjunto de áreas de risco AR juntamente com a quantidade de *UTs* (Unidade de transporte, apresentada na Sessão 2.3.1) que devem ser retiradas da área de risco e o conjunto de abrigos AB com suas respectivas capacidades (lembrando que a capacidade dos abrigos é negativa pois representa quantas UTs ainda cabem no abrigo).

Como foi apresentado na Sessão 2.3.1 todas as medidas de capacidade das vias, quantidade de indivíduos nas áreas de risco e capacidade dos abrigos é medida levando em conta a UT e uma única unidade de tempo.

O método não abrange a etapa de criação do grafo a partir do mapa do local, juntamente com o calculo das capacidades das vias e abrigos e a quantidade de pessoas presentes em cada área de risco. Assumimos que estas informações já foram coletadas e transformadas no grafo G, no conjunto AR e no conjunto AB que irão compor a entrada do método. Apesar do trabalho não considerar a etapa de modelagem do mapa no grafo, no Capítulo 2 é apresentada uma maneira de obter o grafo de um mapa.

Como pode ser visto na Sessão 4.2 o método proposto faz uso do método de Ford-Fulkerson para determinar as rotas que compõe o fluxo máximo, entretanto como foi apresentado na Sessão 3.2 o método de Ford-Fulkerson não comporta várias origens e vários destinos. Para que possamos usá-lo é necessário uma adaptação no grafo de entrada, a adaptação é baseada na técnica apresentada na Sessão 3.1.3.2 para transformar o problema do fluxo máximo com

várias origens e vários destisno no problema do fluxo máximo com uma única origem e um único destino.

Inicialmente são criados 2 novos vértices, o primeiro é uma superorigem S e o segundo um superdestino T. Para cada vértice $s_i \in AR$ é criada uma aresta $(S, s_i) \setminus c(S, s_i) = c_v(s_i)$. Por sua vez, para cada vértice $t_i \in AB$ é criada uma aresta $(t_i, T) \setminus c(t_i, T) = |c_v(t_i)|$. Dessa forma passamos a ter apenas uma única origem S e um único destino T. Perceba que $f(S, s_i) \leq (S, s_i)$ e que $f(t_i, T) \leq (t_i, T)$, com isso podemos inferir que o fluxo que escoa entre a superorigem e todas as áreas de risco não é maior que a quantidade de UT presentes nas áreas de risco, como pode ser visto na Equação 4.1. E que o fluxo que escoa de todos os abrigos em direção ao superdestino não é maior que a capacidade dos abrigos como pode ser visto na Equação 4.2.

$$\sum_{i \in AR} f(S, s_i) \leq \sum_{i \in |AR|} c_v(s_i) \tag{4.1}$$

$$\sum_{i \in |AB|} f(t_i, T) \leq \sum_{i \in |AB|} c_v(t_i) \tag{4.2}$$

Desa forma conseguimos garantir que o fluxo de UTs entre as áreas de risco e os abrigos não vai ser maior que a capacidade dos abrigos. A Figura 15 apresenta um exemplo da adaptação do grafo de entrada.

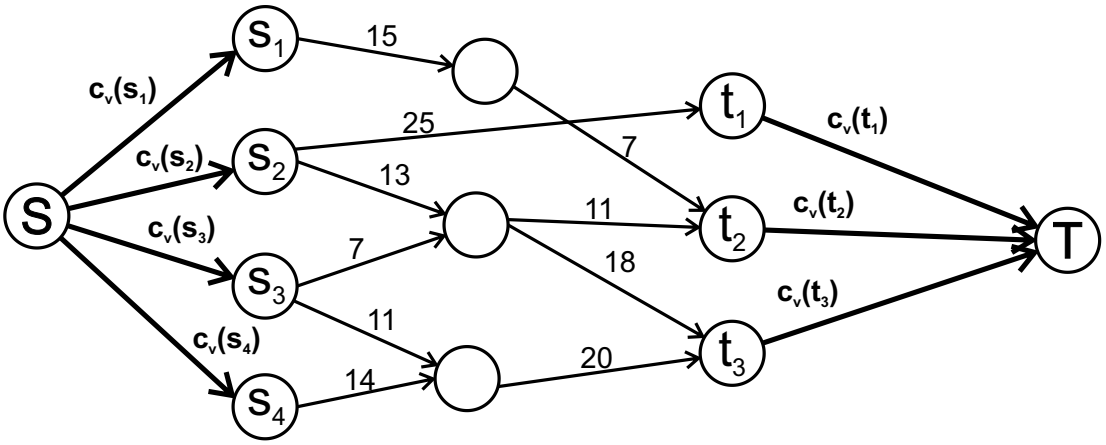

Figura 15 – Inclusão da superorigem e superdestino em uma rede.

Fonte: Autoria própria

Note que as capacidades das vias que ligam a superorigem as áreas de risco tem a capacidade iguais a quantidade de UTs presente em cada área de risco e que a capacidade das vias que ligam cada abrigo ao superdestino é igual ao modulo da capacidade de cada abrigo.

A partir da entrada a próxima sessão apresenta a determinação das rotas de fuga.

4.2 DETERMINAÇÃO DAS ROTAS

Ford-Fulkerson resolve o problema do Fluxo Máximo em redes, sua resposta é o fluxo máximo que pode ser escoado na rede, é possível por meio de adaptações simples guardar quais os caminhos por onde o fluxo deve ser escoado juntamente com a quantidade de fluxo escoado em cada aresta. Executando Ford-Fulkerson tendo como entrada o grafo G definido na Sessão 4.1 a saída é o fluxo máximo de UTs que pode ser escoado pelo grafo G, guardando quais os caminhos foram utilizados pelo método é possível saber então quais rotas devem ser utilizadas para que a quantidade de UTs em movimento seja máxima. Entretanto essas rotas podem não estar de acordo com as restrições estabelecidas na Sessão 2.2, fazendo-se assim necessárias adaptações nas rotas para que estas atendam as restrições propostas.

Primeiramente o método de Ford-Fulkerson é executado para o grafo G. Após a execução de Ford-Fulkerson, para cada vértice presente nas rotas fornecidas é verificado se o mesmo possui mais de um fluxo que incide no vértice e mais de um fluxo que sai do vértice, visto que o relaxamento proposto nas rotas proíbe apenas o caso onde n fluxos incidem em um vértice v_i e n fluxos saem do mesmo vértice v_i. Caso o vértice não atenda a restrição das rotas, a aresta $a_i = (v_1, v_2) \backslash v_1$ e $v_2 \notin \{S, T\}$ por onde passa o menor fluxo é retirada do grafo G. Após a verificação de todos os vértices com a retirada das arestas de menor fluxo nos casos necessários o método de Ford-Fulkerson é executado novamente agora com o grafo $G\,(V, A - \{a_1, a_2, \ldots, a_i\})$.

A execução do método de Ford-Fulkerson seguido da verificação das rotas e remoção de arestas é repetido até que as rotas resultantes da execução do método de Ford-Fulkerson atendam as restrições propostas.

Como a cada iteração do método é retirada uma aresta do grafo, as restrições não proíbem todos os tipos de cruzamento de fluxo e o número de arestas presente no grafo é finito a determinação das rotas tem garantia de parada.

4.3 ESCOAMENTO DO FLUXO

As rotas que foram determinadas na sessão anterior podem não ser capazes de comportar todas as UTs presentes nas áreas de risco, ou seja não é possível que todas as UTs vão para os abrigos ao mesmo tempo, sendo necessário que e evacuação se divida em turnos.

Em cada turno o número de pessoas presente nas áreas de risco e a quantidade de UTs que os abrigos suportam é menor, pois a cada turno UTs saíram das áreas de risco e UTs chegaram aos abrigos, assim se faz necessário que os valores das capacidades das áreas de risco e dos abrigos sejam atualizados a cada turno de evacuação.

A quantidade UTs que saíram da área de risco no turno anterior não está mais na área de risco, dessa forma a nova capacidade é igual a capacidade menos o fluxo total que foi escoado

da área de risco, como é apresentado na Equação 4.3. Da mesma forma o total que UTs que podem ser abrigados em cada abrigo foi diminuido pela quantidade de UTs que já chegaram à esses abrigos, assim temos a Equação 4.4 definindo como as capacidades dos abrigos devem ser atualizadas.

$$c_v(s_i) = c_v(s_i) - \sum_{v \in V(G)} f(s_i, v) \tag{4.3}$$

$$c_v(t_i) = c_v(t_i) + \sum_{v \in V(G)} f(v, t_i) \tag{4.4}$$

Não apenas isso, como já foi dito, as capacidades das arestas que ligam as superorigens às áreas de risco e que ligam os abrigos ao superdestino não deixam que o número de UTs escoado pela pela rede seja maior que a capacidade dos abrigos, assim sendo essas capacidades também devem ser atualizadas. A nova capacidade das arestas que ligam a superorigem às áreas de risco é igual a capacidade da aresta menos o fluxo que foi escoado por ela, como é apresentado na Equação 4.5. Por sua vez, as capacidades das arestas que ligam os abrigos ao superdestino é igual a capacidade da aresta menos o fluxo que foi escoado por ela, conforme pode ser visto na Equação 4.6.

$$c(S, s_i) = c(S, s_i) - f(S, s_i) \tag{4.5}$$

$$c(t_1, T) = c(t_1, T) - f(t_1, T) \tag{4.6}$$

Como todos os valores de capacidade devem ser atualizados, se faz necessário que as rotas sejam recalculadas para o novo turno, e após esse calculo o novo fluxo seja escoado novamente escoado.

Perceba que assim que a capacidade dos abrigos for atingida a capacidade das arestas que ligam os abrigos ao superdistino é atualizada para 0, impedindo que mais UTs sejam evacuadas e finalizando o método. Do mesmo modo, assim que todas as UTs forem evacuadas as capacidades das arestas que ligam a superorigem às áreas de risco serão atualizadas para 0, impedindo assim que mais UTs sejam evacuadas e finalizando o método.

Como o fluxo é limitado pelas capacidades das arestas que ligam a superorigem às áreas de risco e pelas capacidades das arestas que ligam os abrigos ao superdestino, e essas capacidades são diminuídas a cada iteração do método, podemos dizer que a cada iteração o fluxo máximo da rede é menor, ou seja $f_i(S, T) \leq f_i(S, T)$, $i \in \mathbb{Z}$, onde i é a iteração do método.

4.4 CÁLCULO DO TEMPO DE EVACUAÇÃO

Como é descrito na Sessão2.1 a medida de capacidade de uma aresta representa a quantidade de indivíduos, no caso UTs, que atravessam a aresta em uma unidade de tempo. Se o fluxo

máximo da rede é F e este foi calculado a partir de capacidades que representam quantos indivíduos atravessam a aresta em uma unidade de tempo, logo o fluxo máximo representa quantos indivisos atravessam a rede em uma unidade de tempo, assim sendo F indivíduos atravessam essa rede em uma unidade de tempo.

Um fluxo X é calculado em cada turno de evacuação, isso significa que em cada turno de evacuação essas $X \times UTs$ são evacuadas em uma única unidade de tempo, ao seja, cada turno de evacuação representa uma unidade de tempo no tempo total de evacuação. Entretanto como foi apresentado na Sessão 2.1 se faz necessário uma unidade de tempo para que a rede fique totalmente preenchida e possa assim alcançar seu fluxo máximo. Essa unidade de tempo a mais só é aplicada ao primeiro turno, pois a partir do momento do primeiro turno onde a rede se encontrar totalmente preenchida e mesma só ira se esvaziar no final da evacuação.

Dessa forma o tempo total de evacuação é $t + 1$, onde t é a quantidade de turnos de evacuação.

4.5 ALGORITMO

O Algoritmo 4 determina as rotas de fuga apresentado neste capítulo, ele compreende todos os passos descritos nas sessões anteriores. Onde t é o tempo total da evacuação, p é a população evacuada, $c_v(v)$ é a capacidade do vértice v e $c(v_1, v_2)$ é a capacidade da aresta (v_1, v_2).

A etapa de receber a entrada do método apresentada na Sessão 4.1 vai da linha 2 até a linha 9, onde na linha 2 é criada a superorigem, na linha 3 é criado o superdestino, no laço da linha 4 são criadas todas as arestas que ligam a superorigem com as áreas de risco, tendo a capacidade da aresta igual a capacidade da área de rico, por fim no laço da linha 7 as arestas que ligam os abrigos ao superdestino são cridas tendo como capacidade o modulo da capacidade dos abrigos.

O laço da linha 3, é o responsável por determinar quantos turnos de evacuação serão realizados, sendo que a cada turno o laço da linha 14 determina as rotas de fuga que atendem as restrições estabelecidas. O escoamento do fluxo, juntamente com a atualização das capacidades apresentados na Sessão 4.3 são realizados nos laços da linha 18 e 22. Na linha 26 é acrescentada mais uma unidade de tempo ao tempo total da evacuação referente à este turno de evacuação.

A população total evacuada é atualizada na linha 27. Ainda no laço temos a impressão das rotas deste turno seguido da impressão do fluxo obtido neste turno nas linhas 28 e 29 respectivamente.

Por fim temos a impressão do tempo total da evacuação na linha 32, seguido da impressão do tamanho da população evacuada na linha 33.

Foi mencionado na Sessão 4.3 a necessidade de uma unidade de tempo a mais no tempo de evacuação total para que no primeiro turno toda a rede seja preenchida, essa unidade é acres-

Algoritmo 4: ALGORITMO.

Entrada: G, AR, AB

1 **início**
2 Crie a superorigem $S \in G$
3 Crie o superdestino $T \in G$
4 **para** *cada vértice* $s_i \in AR$ **faça**
5 Crie uma aresta $(S, s_i) \setminus c(S, s_i) = c_v(s_i)$
6 **fim**
7 **para** *cada vértice* $t_i \in AB$ **faça**
8 Crie uma aresta $(t_i, T) \setminus c(t_i, T) = c_v(t_i)$
9 **fim**
10 $fx \longleftarrow 1$
11 $t \longleftarrow 1$
12 $p \longleftarrow 0$
13 **enquanto** $fx > 0$ **faça**
14 **enquanto** *rotas não atendem restrições* **faça**
15 $fx \longleftarrow$ fordFulkerson(G, S, T)
16 **fim**
17 **se** $fx > 1$ **então**
18 **para** *cada aresta* (S, s_i) **faça**
19 $c(S, s_i) \longleftarrow c(S, s_i) - f(S, s_i)$
20 $c_v(s_i) \longleftarrow c_v(s_i) - f(S, s_i)$
21 **fim**
22 **para** *cada aresta* (t_i, T) **faça**
23 $c(t_i, T) \longleftarrow c(t_i, T) - f(t_i, T)$
24 $c_v(t_i) \longleftarrow c_v(t_i) - f(t_i, T)$
25 **fim**
26 $t \longleftarrow t + 1$
27 $p \longleftarrow p + fx$
28 Impressão das rotas de fuga dessa leva
29 Impressão do fluxo obtido
30 **fim**
31 **fim**
32 Impressão do tempo total da evacuação
33 impressão da população total evacuada
34 **fim**

Saída: Rotas de fuga por leva, fluxo escoado por leva, rotas e tempo de evacuação

centada na linha 11, onde o tempo de evacuação é iniciado com 1.

Perceba que o algoritmo não impõe de que maneira as arestas que não atendem as restrições das rotas devem ser retiradas, dessa forma cabem diversas implementações nesse processo. Como a avaliação das rotas juntamente com a retirada das arestas que não atendem as restrições das rotas são realizadas em cada iteração do método sua implementações tem influência direta no custo computacional da execução do algoritmo assim como influência na resposta obtida.

4.6 EXEMPLO DE EXECUÇÃO DO ALGORITMO PROPOSTO

Nesta sessão é apresentado um exemplo de execução do método proposto. O método será executado para a o grafo apresentado na Figura 16, nesse grafo temos 14 vértices, 21 arestas, 4 áreas de risco (vértices 1, 2, 4 e 5) com suas respectivas capacidades e 2 abrigos (vértices 12 e 14) também com suas respectivas capacidades.

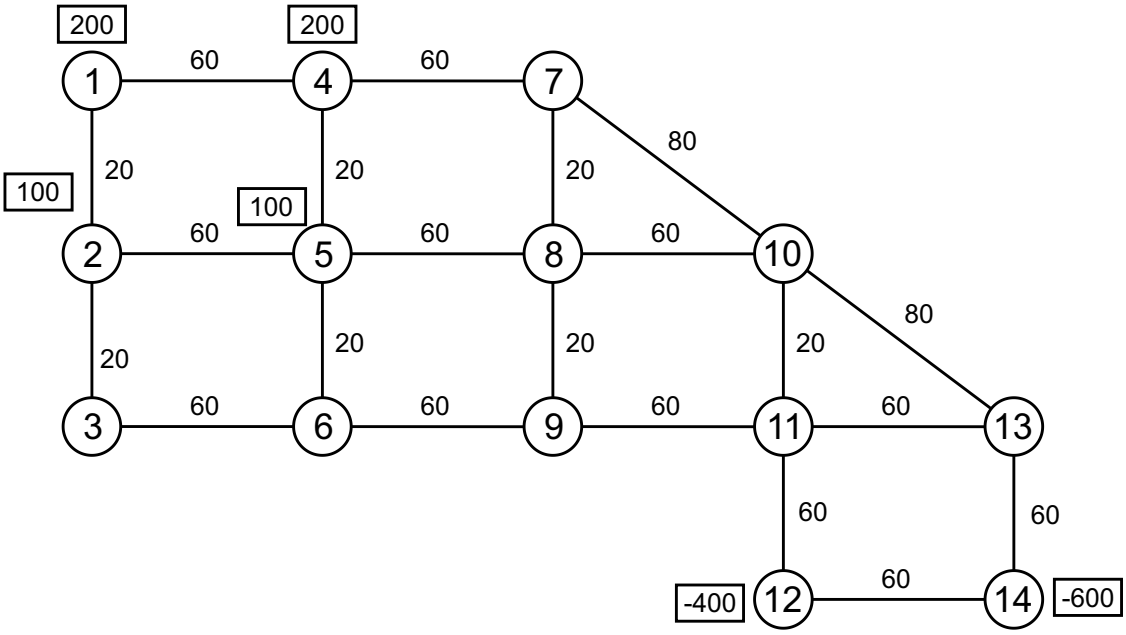

Figura 16 – Exemplo de grafo de entrada.

Fonte: Autoria própria

Seguindo os passos do Algorítmo 4 inicialmente devemos criar a superorigem S e o superdestino T, assim como criar as arestas que ligam a superorigem às áreas de risco e as arestas que ligam os abrigos ao superdestino. A Figura 17 apresenta o grafo de entrada já com os vértices e arestas criados.

Note que as capacidades das arestas que ligam o superdestino às áreas de risco são definidas conforme a Equação 4.1 e as capacidades das arestas que ligam os abrigos ao superdestino são definidas de acordo com a Equação 4.2.

Após a adição dos vértices e arestas verificamos se a capacidade dos abrigos é suficiente para atender a todos, nesse caso temos que a capacidade dos abrigos é

$$c_{ab} = \sum_{t_i \in AB} c_v(t_i) = (-400) + (-600) = -1000$$

, enquanto a capacidae das áreas de risco, ou seja a quantidade de UTs nas áreas de risco é

$$c_{ar} = \sum_{s_i \in AR} c_v(s_i) 200 + 200 + 100 + 100 = 600$$

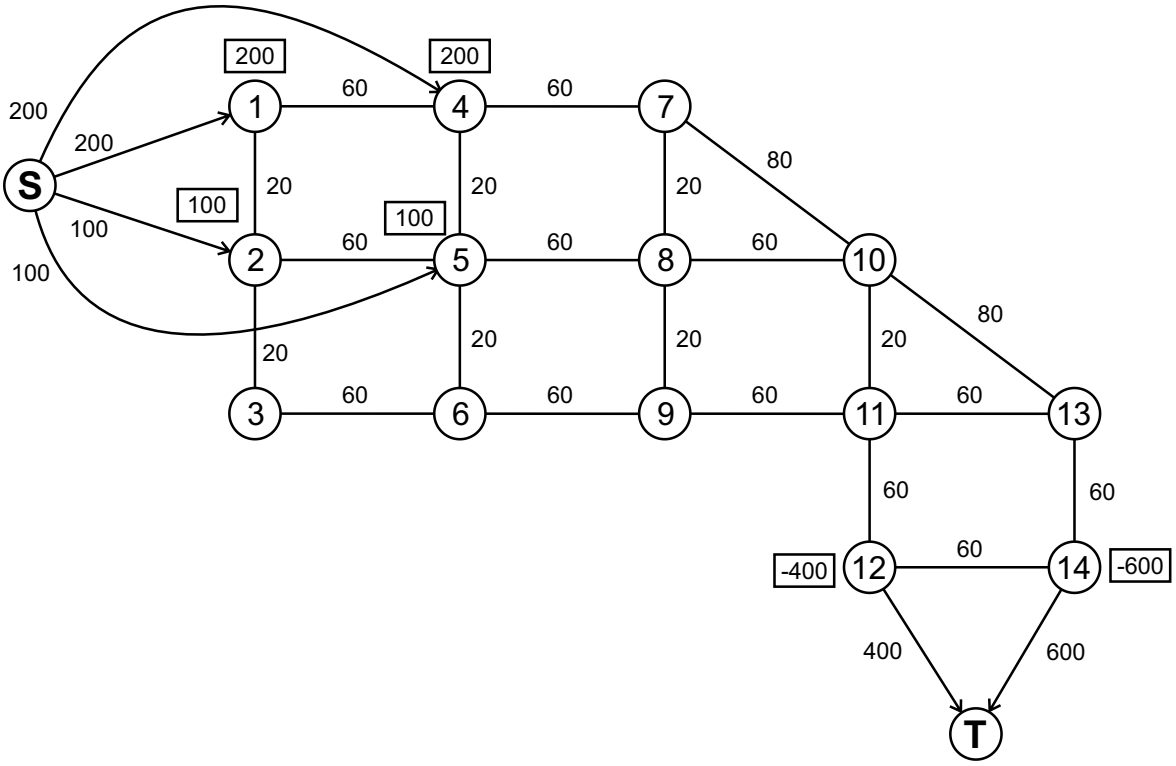

Figura 17 – Exemplo de grafo de entrada com inserção da superorigem e superdestino.

Fonte: Autoria própria

, como $|c_{ab}| = 1000 \geq c_{ar} = 600$ os abrigos tem capacidade suficiente para atender a todas as áreas de risco.

Inicialmente o tempo total de evacuação t é 1 e a população total evacuada p é 0.

Vamos então calcular os fluxos dos turnos de evacuação.

4.6.1 Turno 1

A primeira execução do método de Ford-Fulkerson retorna o fluxo apresentado pela Figura 18. Verificando os caminhos da Figura 18 constatamos que o vértice 10 não está em conformidade com as restrições das rotas estabelecidas na Sessão 2.2, pois existem 2 caminhos chegando no vértice e dois caminhos saindo do vértice, assim temos que remover a aresta por onde passa o menor fluxo no vértice 10, nesse caso removemos a arestas $(8, 10)$, na Figura 18 a aresta removida esta sinalizada com um "X".

Após a remoção da aresta o método de Ford-Fulkerson é executado novamente para que um novo fluxo possa ser calculado, uma vez removida a aresta $(8, 10)$ a Figura 19 apresenta o novo fluxo calculado.

Perceba que a aresta $(8, 10)$ não existe mais, pois foi retirada na verificação das rotas.

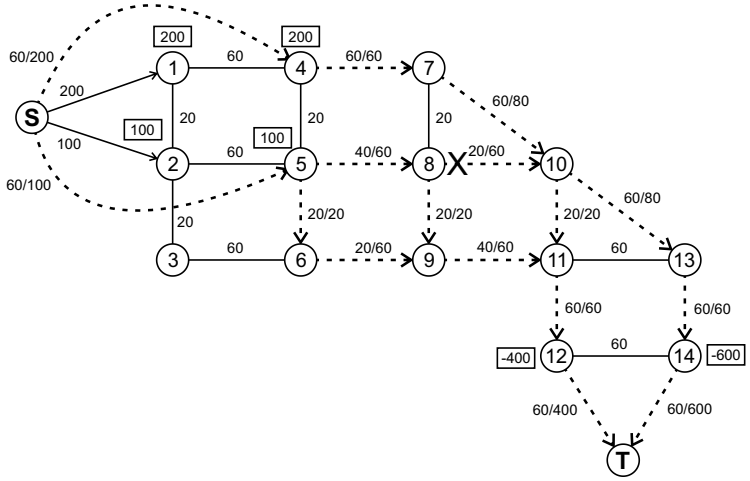

Figura 18 – Turno 1 **- Primeira iteração.**

Fonte: Autoria própria

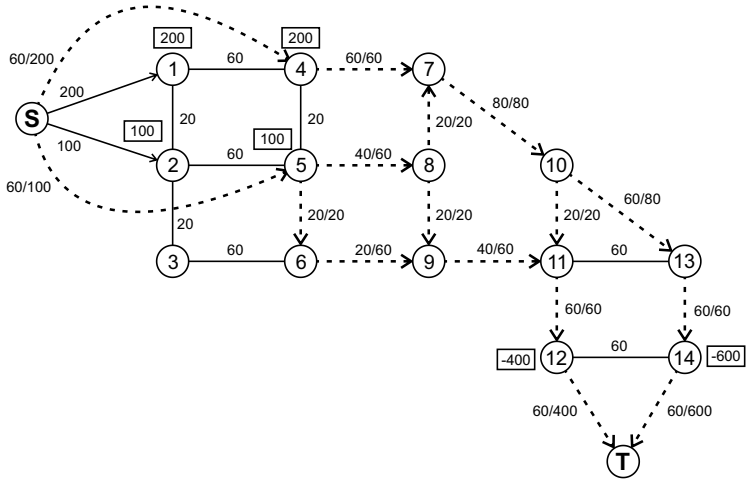

Figura 19 – Turno 1 **- Segunda iteração.**

Fonte: Autoria própria

Como todas as rotas geradas atentem as restrições estabelecidas na Sessão 2.2 nenhuma aresta é removida e o método de Ford-Fulkerson não é executado novamente.

Assim as rotas do turno 1 são: 4 -> 7 (60), 5 -> 6 (20), 5 -> 8 (40), 6 -> 9 (20), 7 -> 10 (80), 8 -> 7 (20), 8 -> 9 (20), 9 -> 11 (40), 10 -> 11 (20), 10 -> 13 (60), 11 -> 12 (60), 13 -> 14 (60).

Ao termino do primeiro turno o fluxo calculado deve ser escoado pela rede e os valores das capacidades devem ser atualizados como foi mostrado na Sessão 4.3, a Figura 20 apresenta o grafo com as capacidades autualizadas e sem a aresta removida.

Além do escoamento do fluxo, o tempo total da evacuação é acrescido de 1 unidade de tempo, ou seja $t = t + 1 = 1 + 1 = 2$ e a população evacuada é acrescida do fluxo evacuado neste turno, $p = p + f(S, T) = 0 + 120 = 120$.

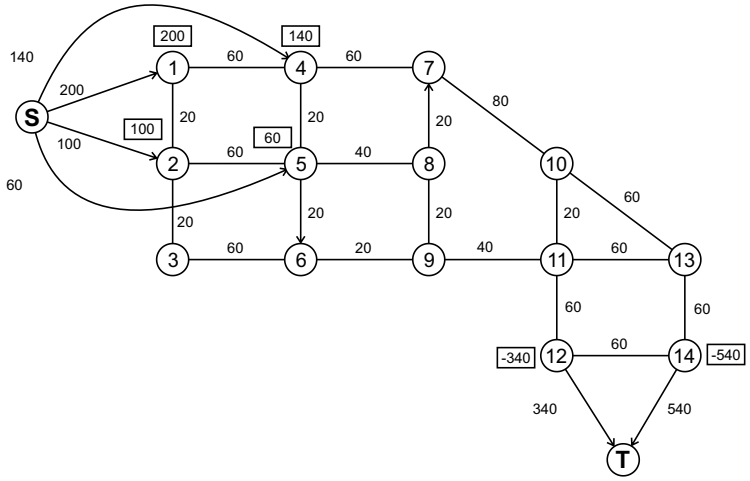

Figura 20 – Turno 1 - Escoamento do fluxo.

Fonte: Autoria própria

Assim encerra-se o primeiro turno.

4.6.2 Turno 2

A execução do método de Ford-Fulkerson resulta nos fluxos apresentados na Figura 21. Perceba que o vértice 11 não está de acordo com as restrições das rotas estabelecidas na Sessão 2.2, dessa forma a resta com menor fluxo que irá ser removida é a aresta $(10, 11)$.

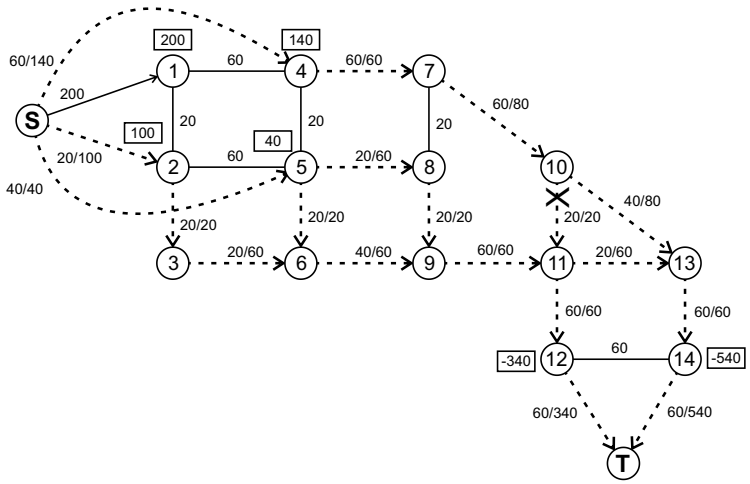

Figura 21 – Turno 2 - Ford-Fulkerson e remoção de aresta.

Fonte: Autoria própria

Após a remoção da aresta o método de Ford-Fulkerson é executado novamente para que um novo fluxo possa ser calculado, uma vez removida a aresta $(10, 11)$ a Figura 22 apresenta o novo fluxo calculado.

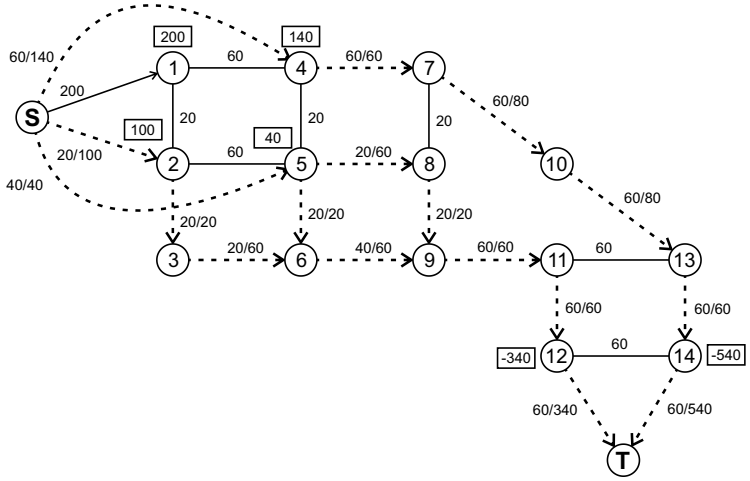

Figura 22 – Turno 2 **- Segundo cálculo do fluxo.**

Fonte: Autoria própria

Como todas as rotas geradas atentem as restrições estabelecidas na Sessão 2.2 nenhuma aresta é removida e o método de Ford-Fulkerson não é executado novamente.

Assim as rotas do turno 2 são: 2 -> 3 (20), 3 -> 6 (20), 4 -> 7 (60), 5 -> 6 (20), 5 -> 8 (20), 6 -> 9 (40), 7 -> 10 (60), 8 -> 9 (20), 9 -> 11 (60), 10 -> 13 (60), 11 -> 12 (60), 13 -> 14 (60).

O fluxo então é escoado e as capacidades recalculadas como apresentado na Sessão 2.1. A Figura 23 apresenta o grafo com as novas capacidades.

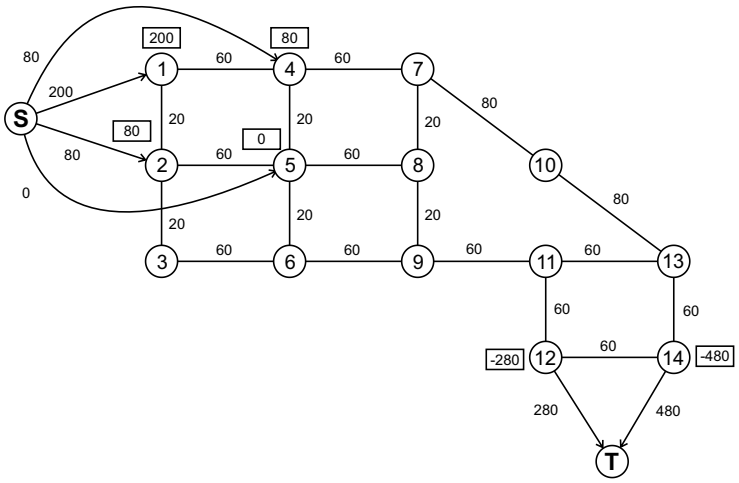

Figura 23 – Turno 2 **- Grafo atualizado.**

Fonte: Autoria própria

O tempo total é atualizado para $t = 3$ e a população evacuada para $p = 240$.

Assim encerra-se o segundo turno.

4.6.3 Turno 3

A execução do método de Ford-Fulkerson resulta nos fluxos apresentados na Figura 24. Como todas as rotas geradas atentem as restrições estabelecidas na Sessão 2.2 nenhuma aresta é removida e o método de Ford-Fulkerson não é executado novamente.

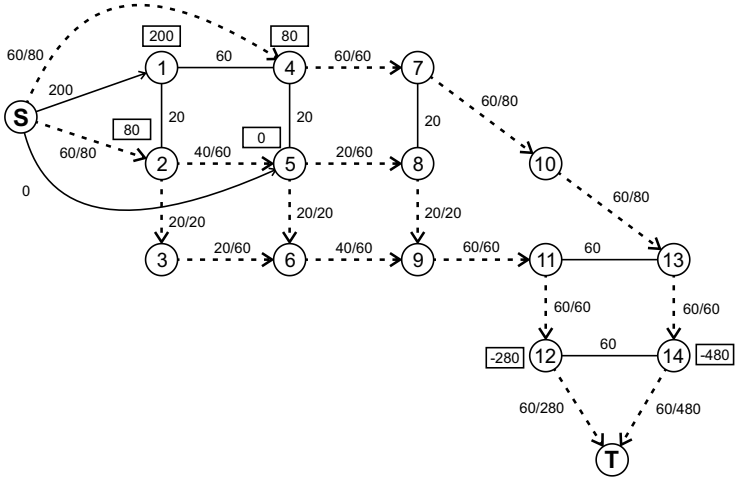

Figura 24 – Turno 3 - Ford-Fulkerson.

Fonte: Autoria própria

Assim as rotas do turno 3 são: 2 -> 3 (20), 2 -> 5 (40), 3 -> 6 (20), 4 -> 7 (60), 5 -> 6 (20), 5 -> 8 (20), 6 -> 9 (40), 7 -> 10 (60), 8 -> 9 (20), 9 -> 11 (60), 10 -> 13 (60), 11 -> 12 (60), 13 -> 14 (60).

O fluxo então é escoado e as capacidades recalculadas como apresentado na Sessão 2.1. A Figura 25 apresenta o grafo com as novas capacidades.

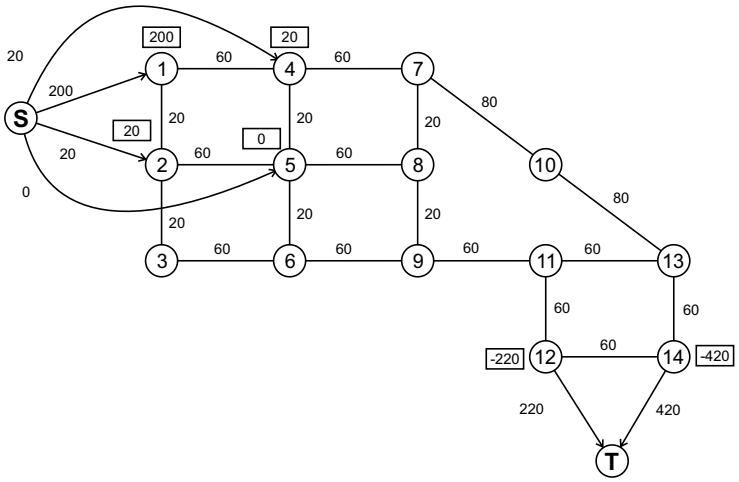

Figura 25 – Turno 3 - Grafo atualizado.

Fonte: Autoria própria

O tempo total é atualizado para $t = 4$ e a população evacuada para $p = 360$.

Assim encerra-se o terceiro turno.

4.6.4 Turno 4

A execução do método de Ford-Fulkerson resulta nos fluxos apresentados na Figura 26. Perceba que o vértice 5 não está de acordo com as restrições das rotas estabelecidas na Sessão 2.2, dessa forma a resta com menor fluxo que irá ser removida é a aresta $(2, 5)$.

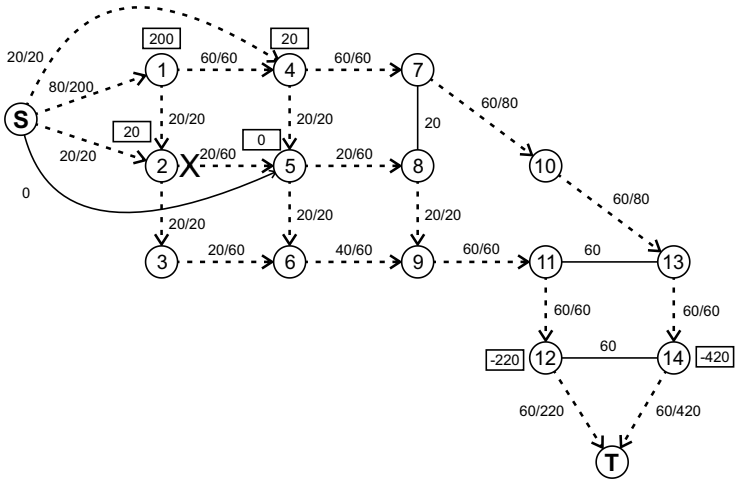

Figura 26 – Turno 4 - Ford-Fulkerson e remoção de aresta.

Fonte: Autoria própria

Após a remoção da aresta o método de Ford-Fulkerson é executado novamente para que um novo fluxo possa ser calculado, uma vez removida a aresta $(2, 5)$ a Figura 27 apresenta o novo fluxo calculado.

Como todas as rotas geradas atentem as restrições estabelecidas na Sessão 2.2 nenhuma aresta é removida e o método de Ford-Fulkerson não é executado novamente.

Assim as rotas do turno 4 são: 1 -> 4 (60), 2 -> 3 (20), 3 -> 6 (20), 4 -> 5 (20), 4 -> 7 (60), 5 -> 6 (20), 6 -> 9 (40), 7 -> 10 (60), 9 -> 11 (40), 10 -> 13 (60), 11 -> 12 (40), 13 -> 14 (60).

O fluxo então é escoado e as capacidades recalculadas como apresentado na Sessão 2.1. A Figura 28 apresenta o grafo com as novas capacidades.

O tempo total é atualizado para $t = 5$ e a população evacuada para $p = 460$.

Assim encerra-se o quarto turno.

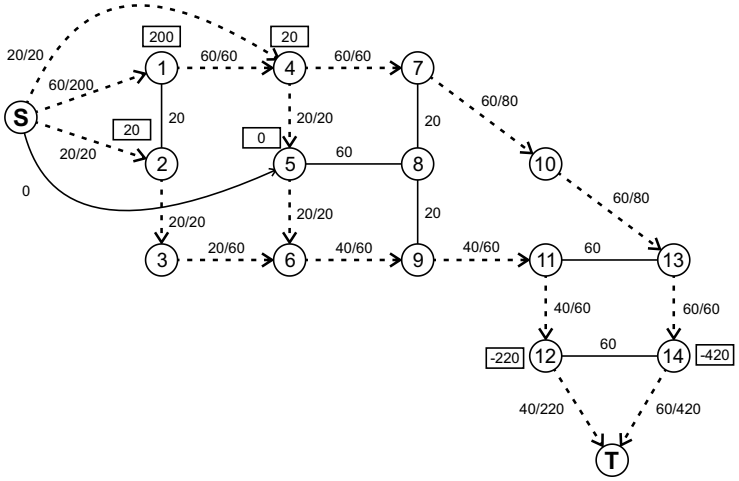

Figura 27 – Turno 4 - Segundo cálculo do fluxo.

Fonte: Autoria própria

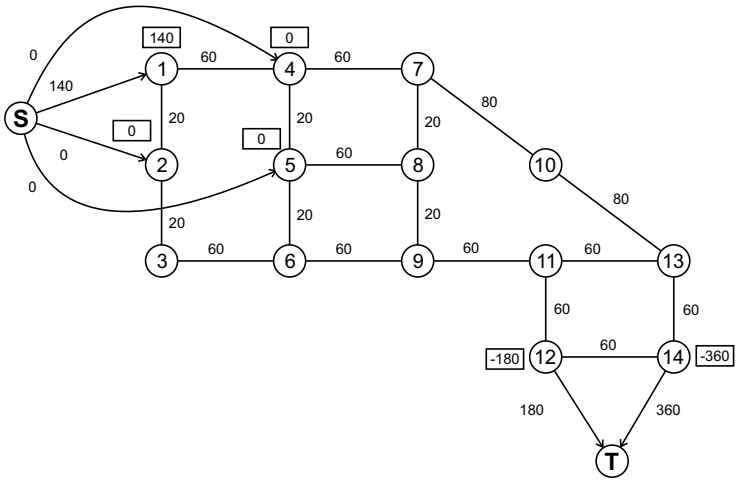

Figura 28 – Turno 4 - Grafo atualizado.

Fonte: Autoria própria

4.6.5 Turno 5

A execução do método de Ford-Fulkerson resulta nos fluxos apresentados na Figura 29. Como todas as rotas geradas atentem as restrições estabelecidas na Sessão 2.2 nenhuma aresta é removida e o método de Ford-Fulkerson não é executado novamente.

Assim as rotas do turno 5 são: 1 -> 2 (20), 1 -> 4 (60), 2 -> 3 (20), 3 -> 6 (20), 4 -> 7 (60), 6 -> 9 (20), 7 -> 10 (60), 9 -> 11 (20), 10 -> 13 (60), 11 -> 12 (20), 13 -> 14 (60).

O fluxo então é escoado e as capacidades recalculadas como apresentado na Sessão 2.1. A Figura 30 apresenta o grafo com as novas capacidades.

O tempo total é atualizado para $t = 6$ e a população evacuada para $p = 540$.

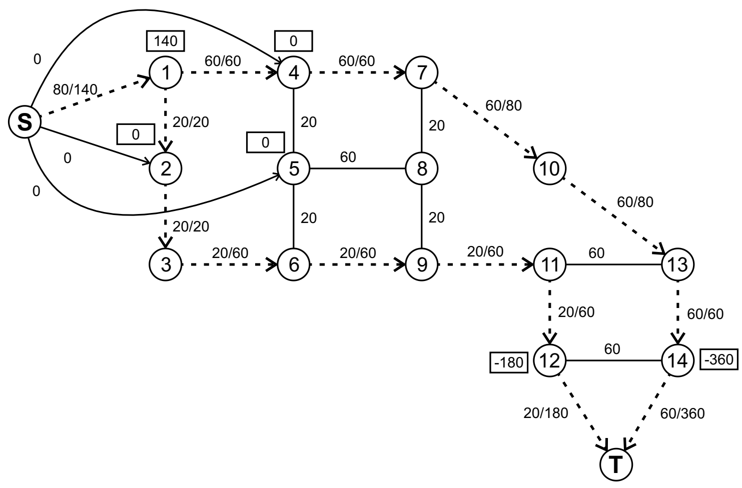

Figura 29 – Turno 5 - Ford-Fulkerson.

Fonte: Autoria própria

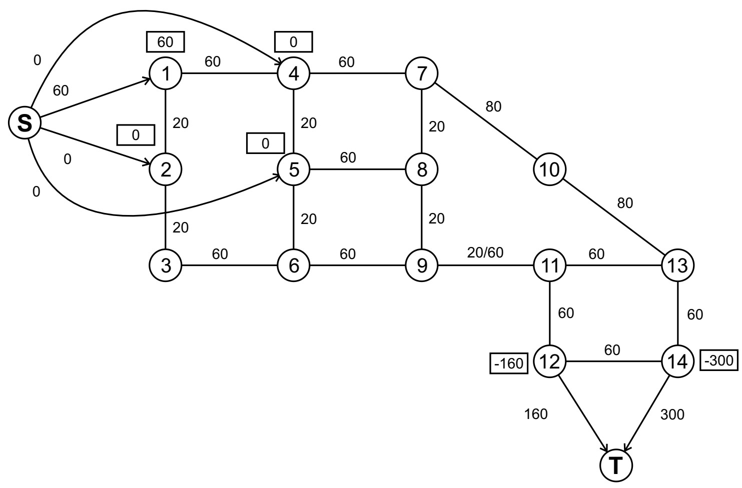

Figura 30 – Turno 5 - Grafo atualizado.

Fonte: Autoria própria

Assim encerra-se o quinto turno.

4.6.6 Turno 6

A execução do método de Ford-Fulkerson resulta nos fluxos apresentados na Figura 31. Como todas as rotas geradas atentem as restrições estabelecidas na Sessão 2.2 nenhuma aresta é removida e o método de Ford-Fulkerson não é executado novamente.

Assim as rotas do turno 6 são: 1 -> 4 (60), 4 -> 7 (60), 7 -> 10 (60), 10 -> 13 (60), 13 -> 14 (60).

O fluxo então é escoado e as capacidades recalculadas como apresentado na Sessão 2.1.

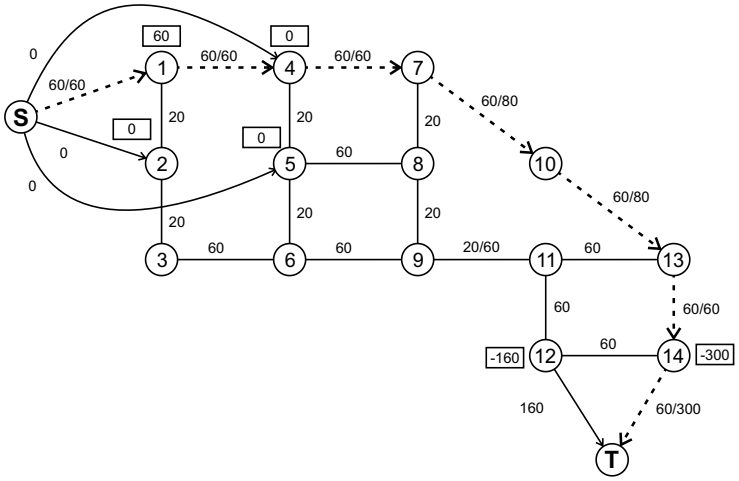

Figura 31 – Turno 6 **- Ford-Fulkerson.**

Fonte: Autoria própria

A Figura 32 apresenta o grafo com as novas capacidades.

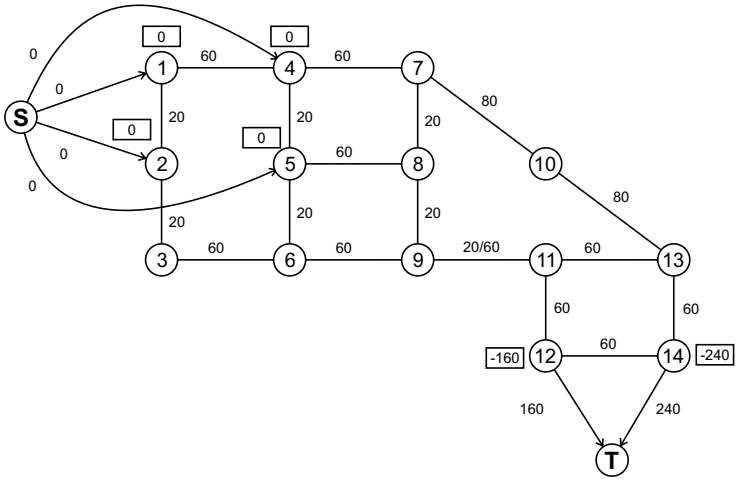

Figura 32 – Turno 6 **- Grafo atualizado.**

Fonte: Autoria própria

O tempo total é atualizado para $t = 7$ e a população evacuada para $p = 600$.

Assim encerra-se o sexto turno.

4.6.7 Turno 7

Como todas as arestas que ligam a superorigem às áreas de risco estão com capacidade 0 o método de Ford-Fulkerson retorno $f(S, T) = 0$, dessa forma não existe mais UTs á serem evacuadas e o turno 7 é encerrado.

4.6.8 Resposta

Por fim temos que $p = 600UTs$ foram evacuada em $t = 7$ unidades de tempo.

4.7 IMPLEMENTAÇÃO DO ALGORITMO PROPOSTO EM LINGUAGEM C

O Algoritmo 4 foi implementado na linguagem de programação C para que pudessem ser realizados testes no mesmo. Nesse programa, o algoritmo de Ford-Fulkerson foi implementado utilizando busca em largura para descobrir novos caminhos aumentantes. Por sua vez as rotas são checadas por meio da verificação de todos os vértices que compõe os caminhos encontrados, de maneira que em cada vértice que não atenda as restrições das rotas a aresta por onde passa a menor quantidade de fluxo é removida do grafo, após a verificação de todos os vértices o algoritmo realiza uma nova iteração até que todos os vértices atendam as restrições de rotas.

O programa em C não conta com um menu de opções, sua única função é determinar as rotas de fuga, dessa forma para utilizá-lo basta inserir as informação referentes a entrada do mesmo e aguardar a resposta.

O código do programa escrito na linguagem C está disponível como Anexo A deste trabalho, também disponibilizamos todos os códigos fontes utilizados, os arquivos de estradas e as saídas em um repositório no Git Hub, esse repositório pode ser acessado pelo link <https://github.com/jonatastbelotti/ModeloAlocacaoFluxo>.

4.7.1 Especificação do formato de entrada dos dados para o algoritmo

A entrada do programa é composta do grafo G onde a evacuação deve ser realizada, juntamente com as áreas de riscos e os abrigos para onde as UTs devem ser levadas.

Primeiramente vem as informações do grafo, a primeira linha é composta pelo número de vértices e arestas do grafo respectivamente. Em seguida as próximas linhas devem ser preenchidas com as arestas do grafo, cada aresta é descrita com 3 números, o primeiro referente à qual vértice a aresta sai, o segundo em qual vértice a aresta incide e o terceiro a capacidade da aresta. A próxima linha informa a quantidade de áreas de risco e de abrigos presente do grafo respectivamente. As próximas linhas descrevem as áreas de risco, cada área de risco é informada com 2 números, o primeiro informa qual o vértice é a área de risco e o segundo informa quantas pessoas devem ser retiradas desse vértice. As próximas linhas descrevem os abrigos, cada abrigo é descrito por 2 números inteiros, o primeiro refente à qual vértice é o abrigo informado e o segundo referente a capacidade desse abrigo (lembrando que a capacidade dos abrigos é negativa pois representa quantas UTs ainda podem ser colocadas nele).

Abaixo apresentamos um exemplo de entrada do programa, essa entrada é referente ao grafo apresentado na Figura do exemplo apresentado na Sessão 4.6.

```
14 21
1 2 20
1 4 60
2 3 20
2 5 60
3 6 60
4 5 20
4 7 60
5 6 20
5 8 60
6 9 60
7 8 20
7 10 80
8 9 20
8 10 60
9 11 60
10 11 20
10 13 80
11 13 60
11 12 60
12 14 60
13 14 60
4 2
1 200
2 100
4 200
5 100
12 -400
14 -600
```

Após a informação da entrada o programa ira processar as rotas e dar a resposta.

4.7.2 Especificação do formato de saída da resposta do algoritmo

A saída do algoritmo é composta pelas rotas de fuga estabelecidas em cada turno de evacuação, juntamente com a população evacuada em cada turno. Após as informações referen-

tes aos turnos é impresso na tela o tempo total da evacuação, juntamente com a população total evacuada.

Caso a capacidade dos abrigos não seja suficiente para abrigar a todos, é impressa uma mensagem avisando quantas UTs poderão ser salvas, as rotas de fuga de cada turno e o tempo de evacuação serão calculados apenas levando em conta o número de UTs que podem ser salvas.

As rotas são apresentadas um caminho por linha, cada caminho é descrito por 3 números, sendo que a seta entre os dois primeiros indica qual o sentido que as UTs devem percorrer e o terceiro número representa quantas UTs devem passar por essa via.

–Rotas do 1º turno————————————
4 -> 7 (60)
5 -> 6 (20)
5 -> 8 (40)
6 -> 9 (20)
7 -> 10 (80)
8 -> 7 (20)
8 -> 9 (20)
9 -> 11 (40)
10 -> 11 (20)
10 -> 13 (60)
11 -> 12 (60)
13 -> 14 (60)
Fluxo: 120

–Rotas do 2º turno————————————
2 -> 3 (20)
3 -> 6 (20)
4 -> 7 (60)
5 -> 6 (20)
5 -> 8 (20)
6 -> 9 (40)
7 -> 10 (60)
8 -> 9 (20)
9 -> 11 (60)
10 -> 13 (60)
11 -> 12 (60)
13 -> 14 (60)
Fluxo: 120

–Rotas do 3º turno————————————

2 -> 3 (20)
2 -> 5 (40)
3 -> 6 (20)
4 -> 7 (60)
5 -> 6 (20)
5 -> 8 (20)
6 -> 9 (40)
7 -> 10 (60)
8 -> 9 (20)
9 -> 11 (60)
10 -> 13 (60)
11 -> 12 (60)
13 -> 14 (60)
Fluxo: 120

–Rotas do 4º turno————————————
1 -> 4 (60)
2 -> 3 (20)
3 -> 6 (20)
4 -> 5 (20)
4 -> 7 (60)
5 -> 6 (20)
6 -> 9 (40)
7 -> 10 (60)
9 -> 11 (40)
10 -> 13 (60)
11 -> 12 (40)
13 -> 14 (60)
Fluxo: 100

–Rotas do 5º turno————————————
1 -> 2 (20)
1 -> 4 (60)
2 -> 3 (20)
3 -> 6 (20)
4 -> 7 (60)
6 -> 9 (20)
7 -> 10 (60)
9 -> 11 (20)

10 -> 13 (60)
11 -> 12 (20)
13 -> 14 (60)
Fluxo: 80

–Rotas do 6º turno————————————
1 -> 4 (60)
4 -> 7 (60)
7 -> 10 (60)
10 -> 13 (60)
13 -> 14 (60)
Fluxo: 60

População evacuada: 600 UTs.
O tempo total para evacuar 100.00 % da população é: 7 unidades de tempo.

5 TESTES E RESULTADOS

Uma vez o modelo criado e implementado, faz-se necessários testes para determinar o quão próximo da resposta ótima ele é capaz de chegar. Neste capitulo serão descritos de que forma os testes foram planejados, realizados e quais os resultados obtidos.

A partir do método criado no Capítulo 4 foram implementadas duas versões do algoritmo:

1. **Aplicando o relaxamento das rotas -** A primeira apresenta a aplicação do relaxamento nas rotas proposto na Sessão 2.2, visando assim incluir um maior número de arestas como arestas possíveis de estrem presentes na solução do problema.

2. **Aplicando rotas totalmente disjuntas -** A segunda faz uso de restrições de rotas mais drásticas, aplicando o conceito de rotas totalmente disjuntas apresentadas na Sessão 3.1.2.

É importante ressaltar que as duas implementações diferem-se apenas no que diz respeito as restrições das rotas, sendo uma, rotas totalmente disjuntas, que se assemelha mais com o modelo proposto por Campos (1997) e a outra rotas parcialmente disjuntas, segundo as definições apresentadas na Sessão 2.2.

A comparação dos resultados das duas implementações permitirá determinar qual restrição de rotas apresenta o melhor resultado e qual tem menor custo computacional, visto que as duas implementações serão executadas para os mesmos casos de teste e nas mesmas condições.

5.1 CASOS DE TESTE

Para que os testes do método implementado fossem realizados, foi necessário criar um gerador de entradas aleatório, no qual a partir das informações da quantidade de vértices, número de arestas, capacidade máxima das arestas do grafo, número de áreas de risco, número de locais seguros, quantidade máxima de UTs em cada área de risco e a capacidade máxima dos abrigos, tem como saída um grafo que é uma entrada válida para o algoritmo. O grafo resultante do gerador de entradas é um grafo que atende as características das entradas do modelo, sendo um grafo não orientado com capacidade nas arestas e peso nos vértices que são áreas de risco ou abrigos. Além disso, para garantir que exista ao menos um caminho entre cada área de risco e um abrigo todas as entradas geradas são grafos onde o subgrafo induzido é conexo.

Para os testes foram criadas 15 instâncias de teste seguindo os critérios a seguir:

- O número de vértices é iniciado em 200, sendo acrescentado de 200 unidades até ser igual a 1000 vértices;

- Para cada faixa de número de vértices (200, 400, 600, 800 e 1000) foram criadas 3 instâncias de teste, variando nas instâncias a densidade do grafo em 25%, 50% e 75%;
- Todos os casos apresentam capacidade máxima das arestas igual a 200;
- Todos os casos de teste tem 50 áreas de risco e 10 abrigos;
- Todos os casos de teste apresentam capacidade máxima de cada abrigo igual a $70000\ UTs$ e número máximo de UTs presentes em cada área de risco igual a 10000.

O Quadro 2 apresenta as especificações de cada caso de teste gerado pelo gerador aleatório de entradas.

Quadro 2 – Especificação dos casos de teste.

Fonte: Autoria própria

#	Num vértices	Densidade	Num arestas	População
1	200	0,25	5000	254550
2	200	0,5	10000	240586
3	200	0,75	15000	235638
4	400	0,25	20000	222381
5	400	0,5	40000	283330
6	400	0,75	60000	239356
7	600	0,25	45000	247180
8	600	0,5	90000	242987
9	600	0,75	135000	273757
10	800	0,25	80000	234839
11	800	0,5	160000	237726
12	800	0,75	240000	269282
13	1000	0,25	125000	254298
14	1000	0,5	250000	249251
15	1000	0,75	375000	245421

No Quadro 2 é apresentado o número de cada caso de teste, a quantidade de vértices que compõe o grafo, a densidade do grafo juntamente com a quantidade de arestas presentes em cada caso de teste e a população total que deve ser evacuada das áreas de risco.

5.2 PARÂMETROS

Para cada caso de teste o valor do fluxo máximo com várias origens e vários destinos foi calculado utilizando a adaptação do método de *Ford Fullkerson* descrito na Sessão 3.1.3.2. Esse valor é importante pois permite comparar o quão próximo da resposta ótima o método é capaz de chegar, visto que o problema de determinar as rotas de fuga para uma população em

situação de catástrofe foi modelado como um problema de fluxo e o método de Ford-Fulkerson é um método exato que apresenta a resposta ótima para o problema do Fluxo Máximo.

Os seguintes parâmetros foram medidos nos testes para os dois algoritmos:

- **Tempo de execução em segundos -** Esse valor possibilita que o custo computacional do método possa ser mensurado, além de permitir a comparação do tempo de execução dos dois algoritmos implementados e juntamente com o número de vértices e arestas presente em cada caso de teste determinar qual variável tem maior influência no tempo de execução do algoritmo.

- **Fluxo máximo -** Como foi mencionado na Sessão 4.3 o fluxo de um turno é sempre menor ou igual que o fluxo do turno anterior, dessa forma o fluxo do primeiro turno é o maior fluxo da rede. Esse valor possibilita juntamente com o fluxo máximo do grafo calculado com o método de *Ford Fullkerson* adaptado verificar o quão próximo da reposta ótima as duas implementações chegaram. Além de juntamente com a medida do tempo de evacuação estabelecer uma relação entre o fluxo máximo escoado pela rede e o tempo de evacuação da população.

- **Número de arestas removidas no primeiro turno -** Esse valor serve para comparação de quantas arestas foram necessárias serem removidas para que o fluxo máximo calculado atenda as restrições estabelecidas.

- **Tempo total de evacuação -** Possibilita verificar qual restrição de rotas apresenta melhor desempenho mediante a resolução do Problema de Evacuar a população o mais rápido possível.

- **Total de arestas removidas ao final da execução -** Esse valor permite comparar quantas arestas tiveram de ser removidas durante toda execução do método para que as rotas atendessem as restrições. Possibilita que seja estabelecida uma relação entre o total de arestas removidas e o tempo de execução do algoritmo, buscando uma comparação dos dois algoritmos.

Para que pudêssemos verificar a quantidade de arestas removidas em cada turno e o total de arestas removidas ao final da execução foram adicionadas *flags* no algoritmo implementado em C.

Os resultados dos testes são apresentados na próxima sessão enquanto as discussões dos resultados são apresentadas na Sessão5.4.

5.3 RESULTADOS

Todos os testes foram realizados em um computador com processador Intel Core i5-2540M de segunda geração com frequência de $2,5$GHz, contando com 4 Gigabytes de memória ram DDR3 667MHz. Sendo executados na distribuição Linux Fedora 22 de 64 bits. No momento dos testes nenhum outro programa exceto o sistema operacional e seus recursos estava em execução.

Todos os casos de teste foram executados até o fim, sendo que toda a população presente nas áreas de risco foi evacuada para os locais seguros em todos os casos de teste, exceto no caso de teste 9 onde a população era de $273757UTs$ mas os abrigos comportavam apenas $241357UTs$, não sendo possível então evacuar toda a população.

O Quadro 3 apresenta os tempos de execução e o tempo total de evacuação do algoritmo que implementa o relaxamento nas rotas e do algoritmo que implementa rotas totalmente disjuntas para cada instância de teste.

Quadro 3 – Tempos de execução e de evacuação obtidos nos casos de teste.

Fonte: Autoria própria.

	Nosso método		**Rotas disjuntas**	
#	**Execução (s)**	**Evacuação**	**Execução (s)**	**Evacuação**
1	8,38	26	2,474	546
2	11,506	25	4,569	629
3	19,027	27	7,036	560
4	105,399	20	31,088	1317
5	133,326	34	62,17	899
6	202,568	31	83,266	1485
7	588,063	16	121,468	889
8	552,867	8	265,364	3352
9	724,748	9 (*)	297,313	3683 (*)
10	1189,123	9	455,088	9014
11	1470,438	7	622,282	1942
12	2422,226	7	876,494	1176
13	2722,581	5	685,924	797
14	3897,306	5	1490,951	4866
15	5829,33	5	1524,549	1165

(*) Não foi possível evacuar todo população, pois a capacidade dos abrigos era insuficiente.

Apenas com os dados do Quadro 3 é possível verificar que o método proposto com o relaxamento das rotas se mostrou mais eficiente para resolver o problema, visto que o tempo de evacuação apresentado por ele é inferior que o algoritmo que implementa rotas totalmente disjuntas em todos os casos de teste. Entretanto discussões mais profundas são apresentadas na Sessão 5.4.

Por sua vez o Quadro 4 apresenta os dados do fluxo máximo escoado em cada caso de teste pelos dois algoritmos, dados estes utilizados para a comparação do relaxamento das restrições das rotas com rotas totalmente disjuntas.

Quadro 4 – Resultados - Fluxo máximo primeiro turno e quantidade de arestas removidas.

Fonte: Autoria própria

	Nosso método		**Rotas disjuntas**	
#	**Fluxo máximo**	**Arestas removidas**	**Fluxo máximo**	**Arestas removidas**
1	28707	2978	1443	3805
2	42967	7267	1699	7031
3	52175	10714	1740	9659
4	49780	12414	1385	15693
5	70527	32216	1619	24181
6	80935	45636	1543	31029
7	78870	32487	1839	31378
8	99841	75481	1479	48242
9	107940	101886	1732	63935
10	96973	50752	1647	51985
11	129768	133268	1683	78110
12	135991	186152	1748	101772
13	125691	81478	1733	73841
14	150148	193255	1623	119776
15	165244	284619	1751	139737

As discussões à respeito dos resultados obtidos são apresentadas na próxima sessão.

5.4 DISCUSSÕES

Nesta sessão são apresentadas as discussões à respeito dos resultados obtidos nos testes.

5.4.1 Fluxo máximo

O gráfico da Figura 33 apresenta os dados referentes ao fluxo máximo obtido pelos dois algoritmos em cada caso de teste. Como mencionado na Sessão 4.3 o fluxo máximo escoado pela rede é o fluxo do primeiro turno, pois a cada iteração as capacidades das arestas que ligam a superorigem às áreas de risco e das arestas que ligam os abrigos ao superdestino são diminuídas pelo fluxo escoado no turno. No gráfico temos o fluxo máximo da rede calculado com o método de Ford-Fulkerson representado pela linha com pontos em forma de círculos, logo a baixo temos o fluxo máximo obtido pelo algoritmo que implementa o relaxamento de rotas representado pela linha com pontos em forma de asteriscos. Por fim o fluxo máximo calculado com o algoritmo

que implementa rotas totalmente disjuntas representado pela linha com pontos em forma de triângulo.

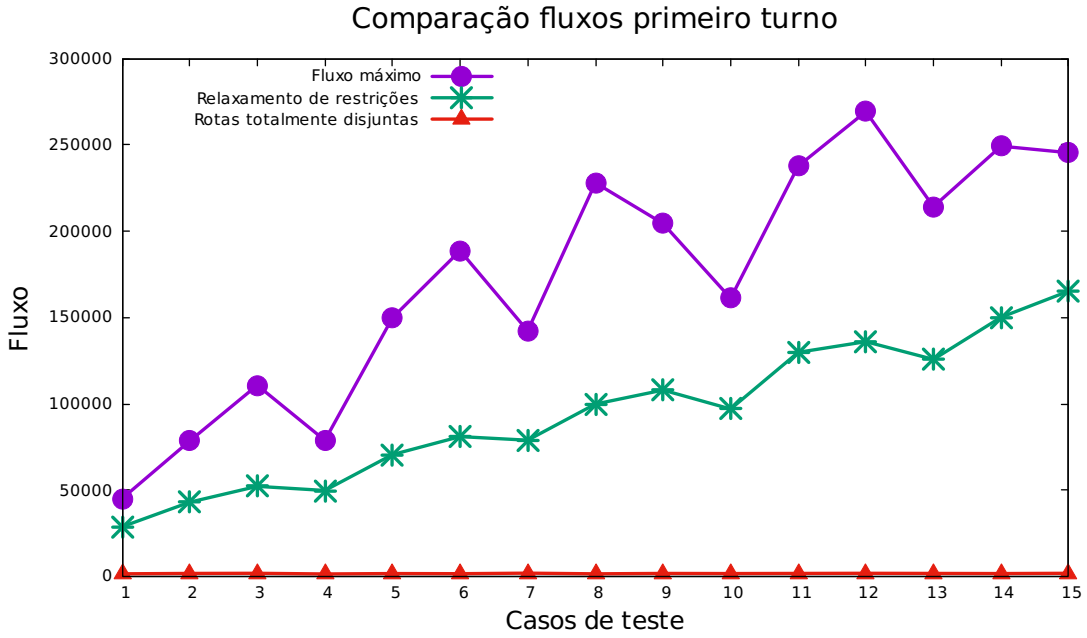

Figura 33 – Comparação dos fluxos máximos com a resposta ótima.

Fonte: Autoria própria.

Note que o fluxo máximo calculado com Ford-Fulkerson é o maior fluxo apresentado no gráfico. O algoritmo que implementa o relaxamento das rotas apresenta o segundo maior fluxo, sendo em média 54% da resposta ótima dada pelo método de Ford-Fulkerson, mesmo com a retirada de arestas o relaxamento de rotas proposto conseguiu resposta que chegam a $67,33\%$ da resposta ótima do problema, no caso de teste 15.

Por sua vez, o fluxo máximo calculado com o algoritmo que implementa rotas totalmente disjuntas apresenta o fluxo mais baixo, sendo em média $1,2\%$ da resposta ótima dada pelo método de Ford-Fulkerson. Isso acontece porque para que as rotas sejam totalmente disjuntas muitas arestas devem ser retiradas, assim, com menos arestas possíveis de estarem na solução do problema o fluxo tende a ser menor.

Para melhor entendimento a Figura 34 apresenta um gráfico com a quantidade de arestas removidas no primeiro turno para cada algoritmo. No gráfico a linha com pontos em forma de círculos representa o método de Ford-Fulkerson, enquanto a linha com pontos em forma de asterisco representa o algoritmo que implementa ao relaxamento de rotas e a linha com pontos em forma de triângulo representa o algoritmo que implementa rotas totalmente disjuntas.

É possível ver que o método de Ford-Fulkerson não removeu nenhuma aresta para calcular o fluxo máximo, ou seja, sua resposta contém rotas sem nenhuma restrição. Mesmo com a restrições de rotas totalmente disjuntas sendo mais drástica que o relaxamento proposto, como pode ser visto no gráfico da Figura 34 o algoritmo que implementa o relaxamento das restrições

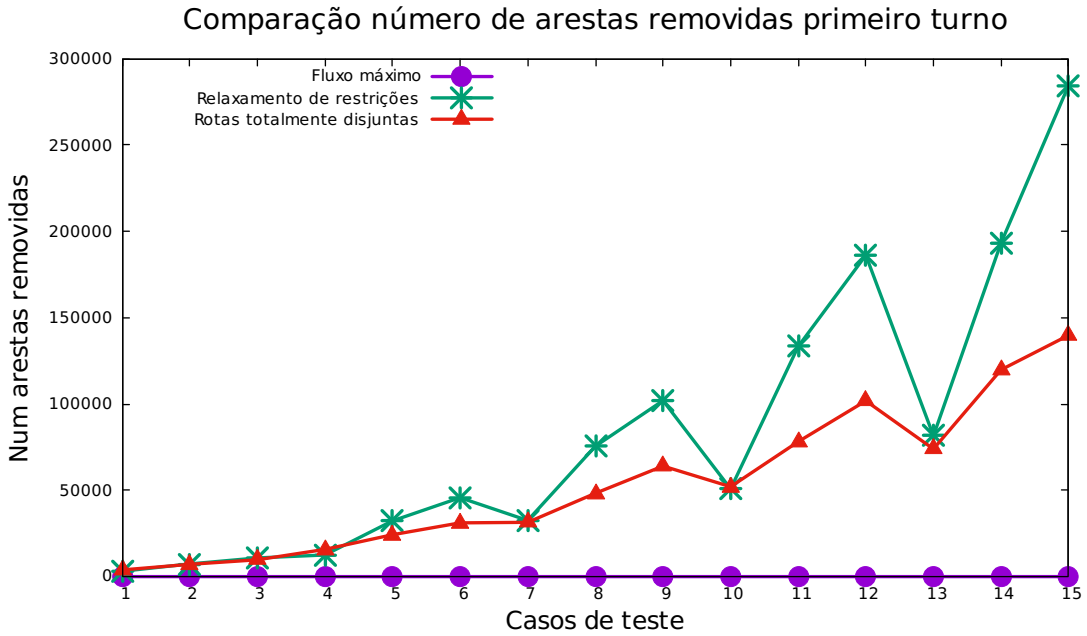

Figura 34 – Número de arestas removidas no primeiro turno.

Fonte: Autoria própria.

foi o que removeu mais arestas ao final do primeiro turno, entretanto mesmo tendo removido mais arestas suas respostas foram melhores, como pode ser visto no gráfico da Figura 33.

Isso se deve ao fato de que rotas totalmente disjuntas comportam um número menor de arestas em sua resposta, visto que nenhum vértice pode estar presente em mais de um caminho. Com uma quantidade menor de arestas presente na resposta, o fluxo escoado tende a ser menor. Em contrapartida, o relaxamento das restrições permite que um maior número de arestas faça parte da solução do problema, fazendo com que o fluxo máximo tenda a ser maior.

O gráfico da Figura 35 apresenta a quantidade de arestas presentes no fluxo máximo encontrado por cada algoritmo e confirma essa informação. Novamente a linha com pontos em forma de círculos representa o método de Ford-Fulkerson, enquanto a linha com pontos em forma de asterisco representa o algoritmo que implementa ao relaxamento de rotas e a linha com pontos em forma de triângulo representa o algoritmo que implementa rotas totalmente disjuntas.

Note que o método de Ford-Fulkerson possui o maior número de arestas presentes no fluxo máximo para todos os casos de teste, seguido do algoritmo que implementa o relaxamento das rotas e por ultimo o algoritmo que implementa rotas totalmente disjuntas.

Comparando os gráficos da Figuras 33 e Figura 35 percebesse que quanto maior o fluxo, maior tende a ser a quantidade de arestas presentes nele.

Como pode ser visto nos gráficos das Figuras 33 e 35, o relaxamento das restrições de rotas possibilita respostas melhores que rotas totalmente disjuntas, pois permite que uma quantidade maior de arestas faça parte da resposta do problema, fazendo assim com que o fluxo na rede tenda a ser maior.

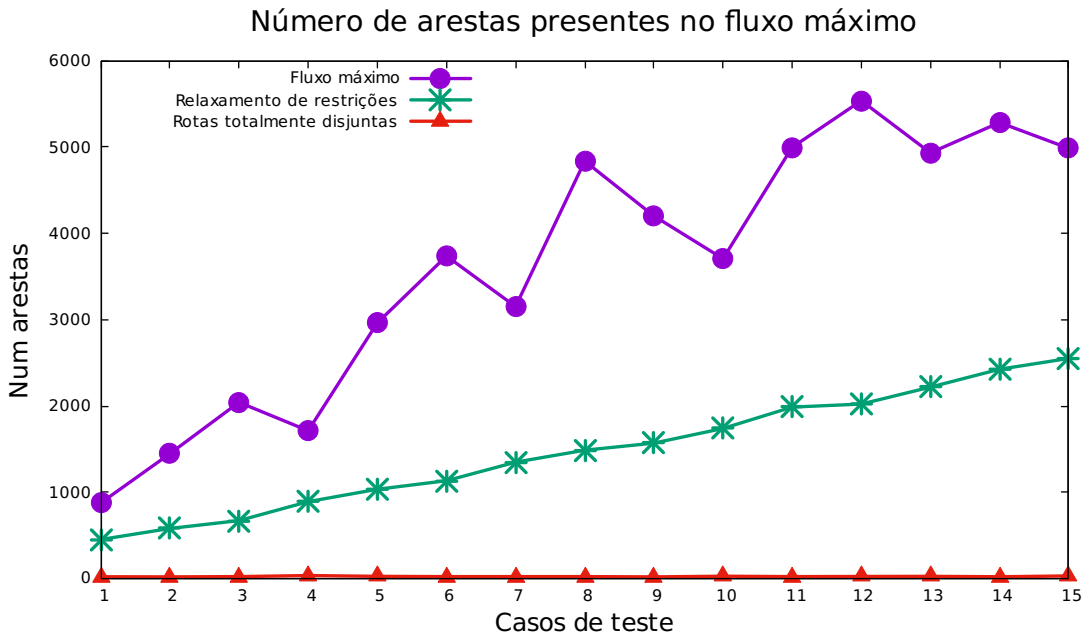

Figura 35 – Número de arestas presentes no fluxo máximo.

Fonte: Autoria própria.

5.4.2 Tempo de evacuação

Como pode ser visto no Quadro 3 o relaxamento de rotas proposto se mostrou mais eficiente do que o uso de rotas totalmente disjuntas no que diz respeito ao tempo de evacuação, a Figura 36 apresenta um gráfico comparativo dos tempos de evacuação. No gráfico a linha com pontos em forma de círculos representa o algoritmo que implementa o relaxamento das rotas e a linha com pontos em forma de triângulos representa o algoritmo que implementa rotas totalmente disjuntas.

Nota-se pelo Quadro 3 e pelo gráfico da Figura 36 que o algoritmo que implementa rotas totalmente disjuntas apresenta tempos de evacuação muito acima do algoritmo que implementa o relaxamento de rotas, visto que o relaxamento de rotas apresenta como maior tempo de evacuação o caso 5 com 34 unidades de tempo, enquanto a implementação de rotas totalmente disjuntas não apresenta tempos de evacuação menores que 546 unidades de tempo em todos os casos de teste.

Isso se deve ao fato de que o algoritmo que implementa o relaxamento de rotas apresenta maiores fluxos, com isso o número de UTs evacuadas a cada turno é sempre maior que o algoritmo que implementa rotas disjuntas. Quanto maior o número de UTs evacuadas por turno, menor o número de turnos necessários para evacuar toda a população, o gráfico da Figura 37 confirma esta informação. No gráfico os pontos em forma de círculo representam o tempo total de evacuação para cada fluxo máximo encontrado em todos os casos de teste.

Percebe-se que conforme o fluxo máximo aumenta no eixo x o tempo total de evacuação

Figura 36 – Gráfico comparação tempo de evacuação.

Fonte: Autoria própria.

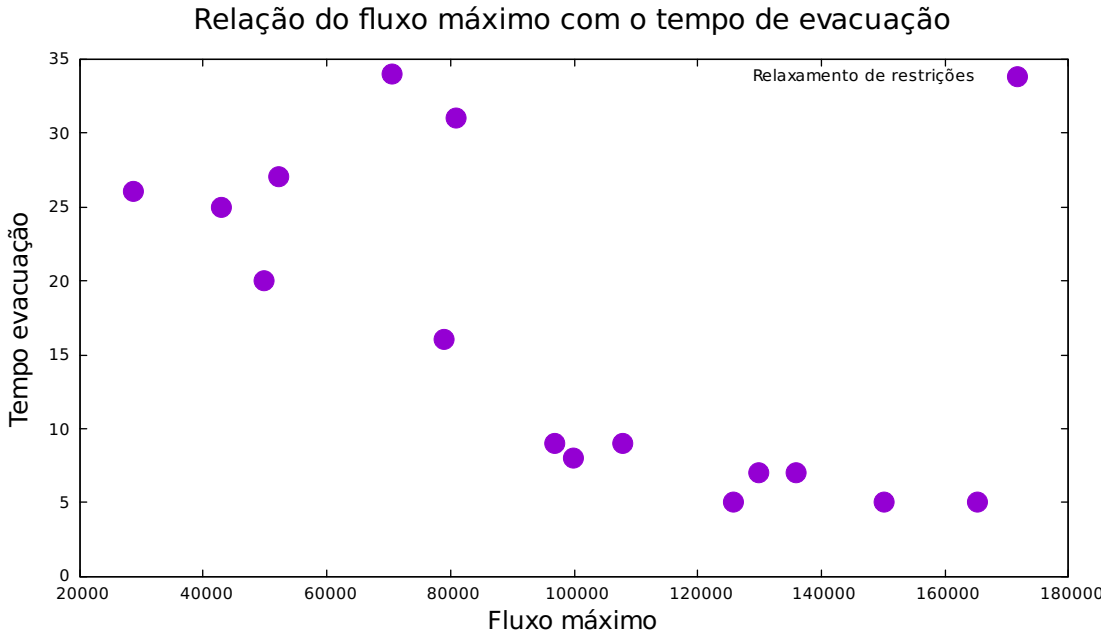

Figura 37 – Relação do fluxo máximo com o tempo de evacuação.

Fonte: Autoria própria.

tende a diminuir no eixo y. Estabelecendo assim uma relação entre o fluxo máximo da rede com o tempo de evacuação da população.

5.4.3 Tempo de execução

O gráfico da Figura 38 apresenta os tempos de execução obtidos pelos dois algoritmos. No gráfico a linha com pontos em forma de círculos representa o algoritmo que implementa o relaxamento das rotas, enquanto a linha com pontos em forma de triângulos representa o algoritmo que implementa rotas totalmente disjuntas.

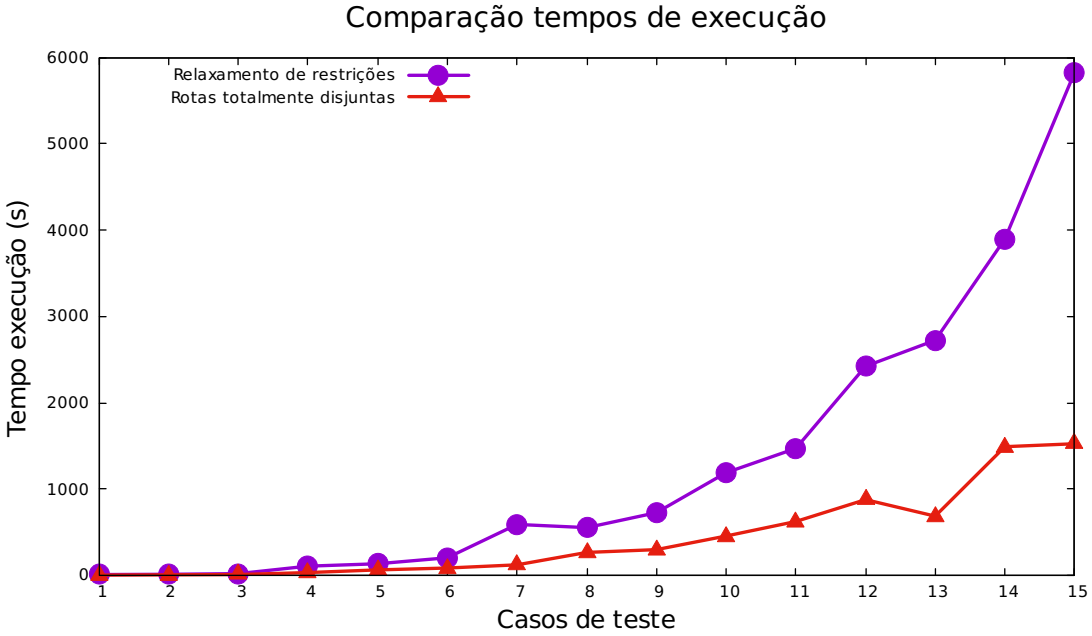

Figura 38 – Gráfico de comparação dos tempos de execução.

Fonte: Autoria própria.

Note que em todos os casos de teste a implementação de rotas totalmente disjuntas obteve tempos de execução menores que o relaxamento de rotas proposto, entretanto como apresenta a Sessão 5.4.2 seus tempos de evacuação da população são sempre piores.

Como apresentado na Sessão 4.2 o método proposto realiza várias iterações do método de Ford-Fulkerson retirando arestas até que as rotas atendam as restrições propostas, dessa forma quanto mais arestas forem removidas, mais vezes o método de Ford-Fulkerson será executado, resultando em um maior tempo de execução do algoritmo. Os gráficos da Figura 39 apresentam a relação entre o total de arestas removidas ao final da execução dos algoritmos com o tempo de execução dos algoritmos. No gráfico da esquerda temos a relação para o algoritmo que implementa o relaxamento das rotas, enquanto o gráfico da direita apresenta a relação para o algoritmo que implementa rotas totalmente disjuntas.

Note que em ambos os gráficos, conforme a quantidade de arestas removidas ao final do algoritmo cresce no eixo x o tempo de execução do algoritmo tende a crescer no eixo y. Dessa forma comprovamos que quanto maior o número de arestas removidas pelo método mais tempo ele levará para ser executado.

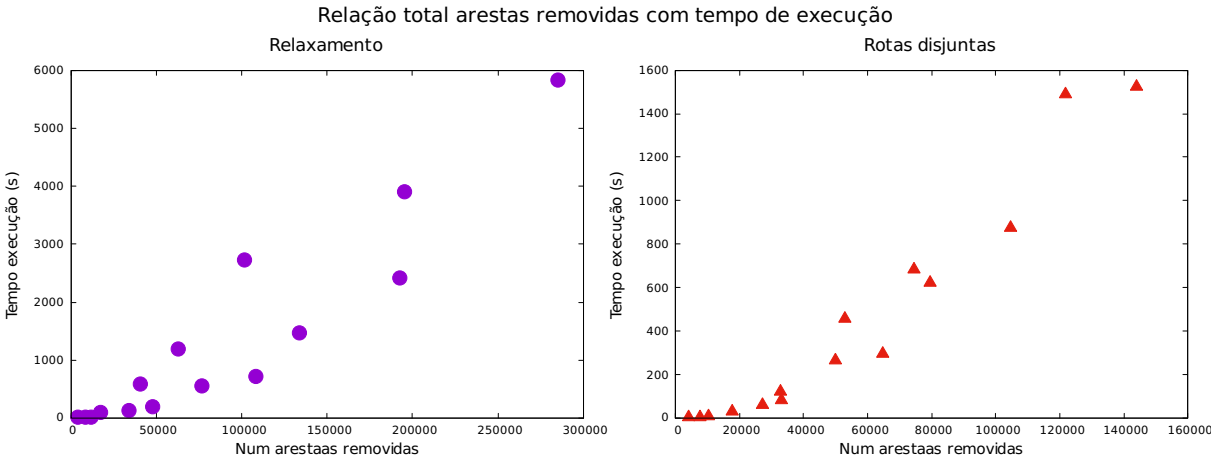

Figura 39 – Relação do total de arestas removidas com o tempo de execução do algoritmo.

Fonte: Autoria própria.

Além do número total de arestas removidas, existem outras variáveis que também influência no tempo de execução do algoritmo, como o número de vértices do grafo, o número de arestas e o fluxo máximo obtido. Com fim de possibilitar uma comparação do peso de cada variável no tempo de execução do método os gráficos da Figura 40 apresentam a relação existente entre o número de vértices do grafo, do número de arestas do grafo, do fluxo máximo da rede e do total de arestas removidas com o tempo total de execução.

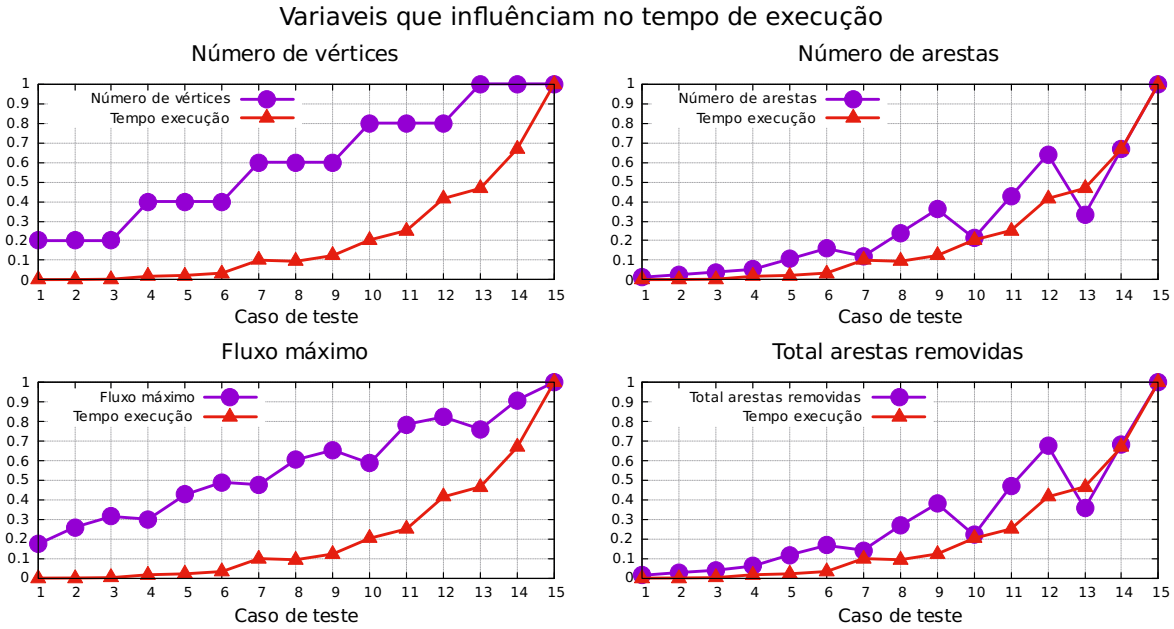

Figura 40 – Variáveis.

Fonte: Autoria própria.

Para que possamos saber quanto o tempo aumentaria se cada variável fosse aumentada, os gráficos foram construídos da seguinte forma: o tempo máximo de execução do algoritmo

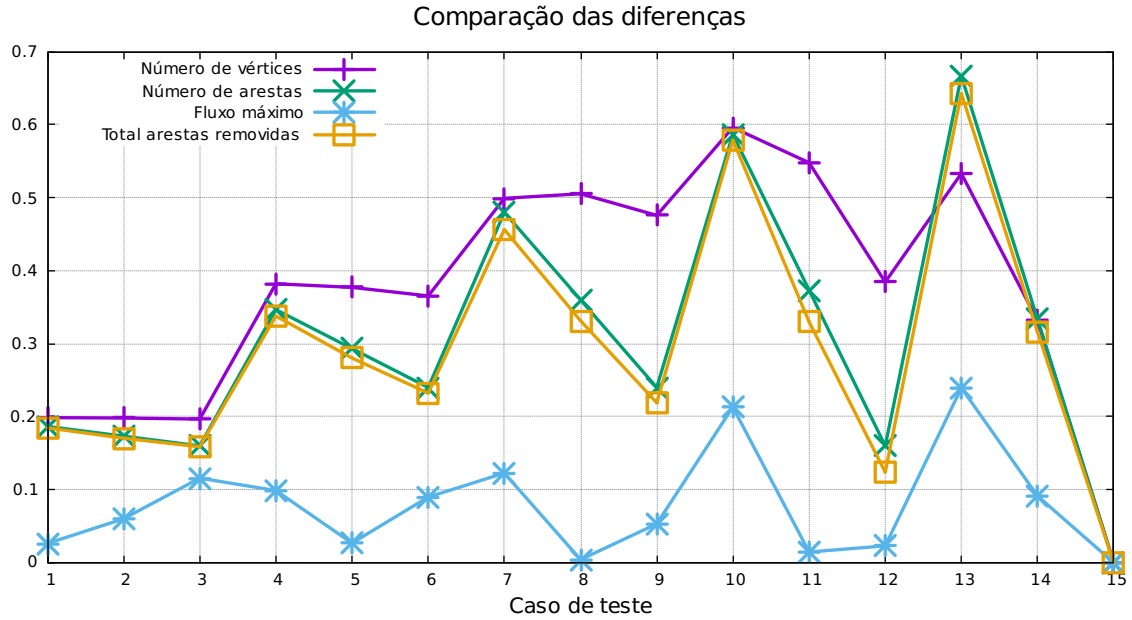

Figura 41 – Diferença das curvas dos gráficos da Figura 40 com a curva do tempo de execução.

Fonte: Autoria própria.

para todos os casos de teste é conhecido, sendo o caso 15 com tempo de execução de $5829,33$ segundos, também é conhecido o maior número de vértices apresentado pelos testes, casos 13, 14 e 15 com 1000 vértices, conhecemos também qual o maior número de arestas dentre todos os casos de teste, caso 15 com 3750000 arestas e conhecemos em qual caso de teste houve o maior número de arestas removidas, caso 15 com 285071 arestas removidas. A partir desses valores, para cada caso de teste calculamos quantos per cento do número máximo de vértices seu número de vértices representa, fizemos o mesmo para o número de arestas, fluxo máximo e e total de arestas removidas, além de calcular quantos per cento do tempo de execução máximo seu tempo de execução representa.

Desse modo para verificar qual variável tem mais influência no tempo de execução do algoritmo basta determinar qual das curvas é mais próxima da curva apresentada pelo tempo de execução. Para facilitar esse processo, no gráfico da Figura 41 são apresentadas essas diferenças já calculadas.

Observando o gráfico da Figura 41 verificamos que a quantidade de arestas da rede tem mais influência no tempo de execução do mesmo que o número de vértices, pois os valores referentes ao número de arestas estão em 14 dos 15 casos de teste abaixo dos valores referentes ao número de vértices.

Também é possível observar que o número de vértices do grafo é a variável que tem menor influencia no tempo de execução do algoritmo.

É evidente ao observar o grafo que o valor do fluxo máximo encontrado pelo algoritmo na rede é a variável que tem maior influência no tempo de execução do algoritmo. Como foi

apresentado na Sessão 3.2.5, a complexidade do método de Ford-Fulkerson é $O(|A|f)$, onde f é o valor do fluxo máximo. Entretanto observando o gráfico notamos que no algoritmo proposto o total de arestas removidas tem mais influência no tempo de execução do algoritmo que o número de arestas.

5.4.4 Total de arestas removidas

Como já foi dito na Sessão 5.4.1 rotas totalmente disjuntas é um tipo de restrição mais severa, permitindo que uma quantidade menor de arestas faça parte da solução do problema, enquanto que o relaxamento de rotas disjuntas é uma restrição mais branda permitindo que um número maior de arestas faça parte da solução do problema. Partindo desse conceito, imagina-se que o algoritmo que implementa rotas totalmente disjuntas remova uma quantidade maior de arestas do grafo ao final da sua execução que o algoritmo que implementa o relaxamento das restrições. Entretanto como pode ser visto no gráfico da Figura 42 não é isso o que acontece.

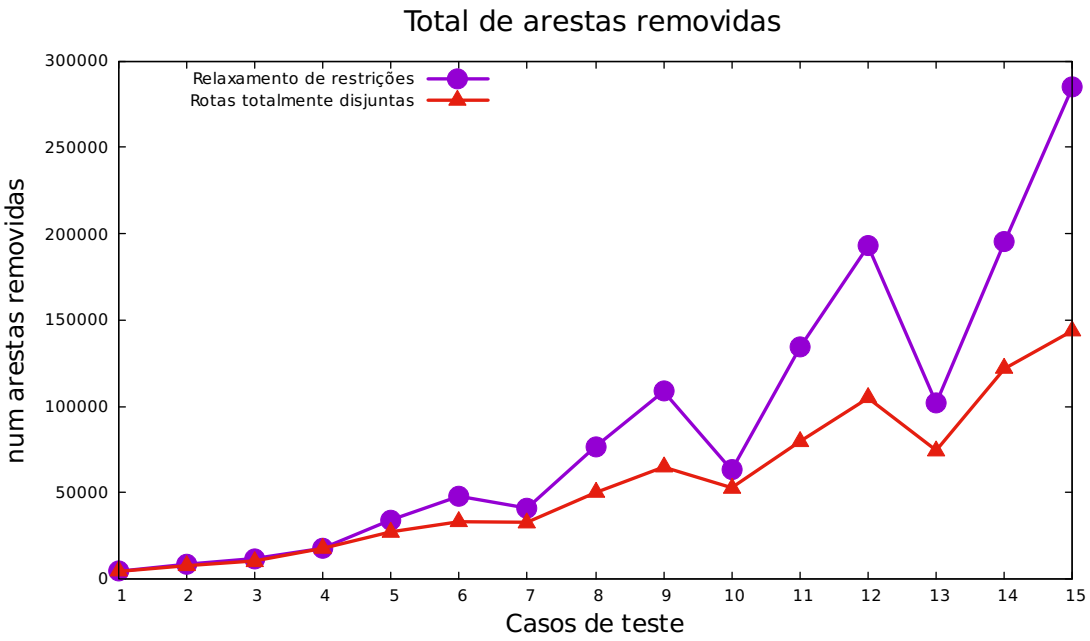

Figura 42 – Número de arestas removidas ao final da execução dos algoritmos.

Fonte: Autoria própria.

Pelos dados do gráfico é possível observar que em todos os casos de teste o algoritmo que implementa o relaxamento das restrições apresenta um maior número de arestas removidas ao final da sua execução.

Para entender o porque isso acontece basta relembrarmos que rotas disjuntas não permitem que um vértice faça parte de mais de um caminho, resultando assim em um menor número de vértices presentes na solução do problema, com um menor número de vértices o número de

arestas também tende a diminuir. Enquanto que o relaxamento das restrições permite que um mesmo vértice faça parte de mais de um caminho, desde que não exista mais de um caminho que chegue no vértice e mais de um caminho que saia do vértice, resultando assim em um maior número de vértices presentes na solução do problema, com um maior número de vértices o número de arestas na solução tende a ser maior. O gráfico da Figura 43 confirma essa informação, nele são apresentados o número de vértices diferentes que aparecem nas rotas propostas pelos dois algoritmos. No gráfico a linha com pontos em forma de círculos representa o número de vértices presentes nas rotas propostas pelo algoritmo que implementa o relaxamento de rotas, enquanto a linha com pontos em forma de triângulos representa o número de vértices presentes nas rotas propostas pelo algoritmo que implementa rotas totalmente disjuntas.

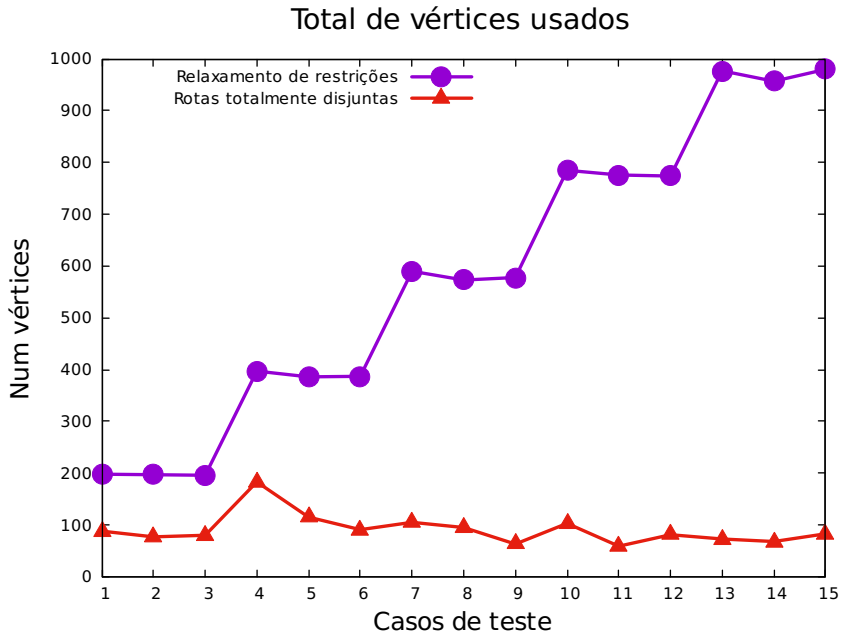

Figura 43 – Total de vértices presentes nas rotas propostas pelos algoritmos.

Fonte: Autoria própria.

Como pode ser observado no gráfico, em todos os casos de teste as rotas propostas pelo algoritmo que implementa o relaxamento de restrições apresentam um maior número de vértices que as rotas propostas pelo algoritmo que implementa rotas totalmente disjuntas.

O método proposto verifica para cada vértice das rotas calculadas com Ford-Fulkerson se o mesmo obedece as restrições de rotas, removendo as arestas que não estão de acordo com as restrições. Com rotas totalmente disjuntas a cada verificação das rotas um número menor de vértices faz parte das rotas, dessa forma nas próximas verificações o número de vértices verificados será menor, fazendo com que o número de arestas que tenham de ser removidas tenda a diminuir. Por sua vez, o relaxamento das restrições possibilita que ao final de cada verificação das rotas um maior número de vértices ainda faça parte das rotas, o que implica que nas próximas verificações um maior número de vértices ainda tenham de ser verificados fazendo com que o número de arestas que tenham de ser removidas tenda a ser maior.

6 TRABALHOS FUTUROS

O método de alocação de fluxo em redes para evacuação de populações apresentado por este trabalho assume que o mapa da região onde a catástrofe ocorreu ou está ocorrendo já foi transformado no grafo de entrada do algoritmo, também assumimos que os valores das capacidades das arestas quantidade de UTs presente em cada área de risco e a capacidade dos abrigos já foi calculada e que todos os valores foram calculados levando em conta a Unidade de Transporte (UT) definida na Sessão 2.3.1. Deixando assim espaço para o desenvolvimento de uma ferramenta que automatize o processo de transformação do mapa em grafo, para a criação da entrada do algoritmo. Tal ferramenta poderia receber uma imagem do mapa da área onde a catástrofe esta ocorrendo e dada uma marcação das áreas de risco e dos locais seguros, por meio de técnicas de processamento de imagens crie o grafo de entrada do algoritmo.

Na Sessão 2.3.1 foi definido a utilização de uma Unidade de Transporte visando restringir o escopo do problema para uma primeira resolução, mediante a primeira abordagem do problema apresentada por este trabalho pode-se estender o método criado para que este tenha suporte a várias unidades de transporte com capacidades e propriedades diferentes, tornando o método mais próximo da realidade.

Como foi apresentado na Sessão 4.5 o método desenvolvido não restringe a forma como as rotas serão checadas e as arestas removidas, visto que essa etapa tem influência direta no custo computacional do algoritmo já que quanto mais eficiente for a maneira como as rotas são checadas e as arestas removidas menor é o número de vezes que o método de Ford-Fulkerson deve ser executado resultando em um tempo de execução menor, além de que a maneira como as arestas são removidas influência direto no fluxo máximo obtido pelo algoritmo, dessa forma uma técnica que escolha as arestas a serem removidas visando um melhor aproveitamento das arestas tem impacto direto no tempo total de evacuação da população.

Existem situações de catástrofes onde infelizmente não é possível salvar todas as pessoas nas áreas de risco (caso de teste 9), o método apresentado por este trabalho não é capaz de determinar o que essas pessoas devem fazer nessas situações, atualmente quando os abrigos não tem capaciidade suficente para atender a todos o método deixa essas pessoas nas áreas de risco. Sendo necessário um estudo mais aprofundado desse tipo de situação, visando determinar de que maneira o método deveria se comportar nessas situações.

O relaxamento de rotas totalmente disjuntas mostrou-se mais eficiente na solução do problema, entretanto ele ainda não é a resposta ótima para o problema, dessa forma ainda é necessário encontrar quais as restrições de rotas mais adequadas, visando assim um maior fluxo de forma a minimizar os possíveis conflitos da população em fuga.

7 CONCLUSÃO

Ao desenvolver esse trabalho nosso objetivo era modelar o problema de determinar as rotas de fuga para uma população em situação de catástrofe como um problema de fluxo em rede, visando um método que possibilitasse o escoamento da maior população possível pela rede ao mesmo tempo. Além disso, buscávamos aplicar um relaxamento sobre as restrições de rotas pleiteando uma melhor solução do problema.

Analisando detalhadamente o problema verificamos que algumas restrições se faziam necessárias para possibilitar uma primeira abordagem, dessa forma introduzimos o conceito de UT.

Dessa forma concluímos que o problema de determinar as rotas de fuga para uma população em situação de catástrofe pode ser modelado como um problema de fluxo, além de propormos um método que resolve o problema e uma implementação para esse método. O algoritmo desenvolvido mostrou-se eficiente na solução do problema. para o problema, solução está que se demonstrou eficiente no que diz respeito da evacuação da população.

REFERÊNCIAS

BBC-BRASIL. **Enchentes causam mortes e transtornos na Europa Central**. 2013. Disponível em: <http://www.bbc.co.uk/portuguese/videos_e_fotos/2013/06/130607_aprenda_enchentes>. Acesso em: 01 dez. 2014.

CAMPONOGARA, E. Métodos de otimização: Teoria e prática. **Florianópolis: Universidade Federal de Santa Catarina**, 2005.

CAMPOS, V. B. G. **Método de Alocação de Fluxo no Planejamento de Transportes em Situação de Emergência: Definição de Rotas Disjuntas**. Tese (Doutorado) — UFRJ, 1997.

CASSOL, V. J. *et al.* Crowdsim: Uma ferramenta desenvolvida para simulação de multidões. In: **Proceedings of the Eleventh Brazilian Symposium on Computer Games and Digital Entertainment (SBGames' 12), Sociedade Brasileira de Computação, SBC**. [S.l.: s.n.], 2012. p. 1–4.

CORMEN, T. *et al.* **Algoritmos Teoria e Prática. Tradução da segunda edição [americana] por Vandberg de Souza**. [S.l.]: Elsevier, Rio de Janeiro, 2002.

DEB, K. *et al.* A fast and elitist multiobjective genetic algorithm: Nsga-ii. **Evolutionary Computation, IEEE Transactions on**, IEEE, v. 6, n. 2, p. 182–197, 2002.

DINITS, E. A. Algorithm of solution to problem of maximum flow in network with power estimates. **Doklady Akademii Nauk SSSR**, MEZHDUNARODNAYA KNIGA 39 DIMITROVA UL., 113095 MOSCOW, RUSSIA, v. 194, n. 4, p. 754, 1970.

EDMONDS, J.; KARP, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. **Journal of the ACM (JACM)**, ACM, v. 19, n. 2, p. 248–264, 1972.

FEOFILOFF, P. Fluxo em redes. **Departamento de Ciência da Computação e Instituto de Matemática e Estatística, São Paulo**, 2004.

GOLDBERG, A. V. **A new max-flow algorithm**. [S.l.]: Laboratory for Computer Science, Massachusetts Institute of Technology, 1985.

HURRICANES. **1970- The Great Bhola Cyclone**. 2005. Disponível em: <http://www.hurricanescience.org/history/storms/1970s/greatbhola>. Acesso em: 01 dez. 2014.

HUTCHINSON, B. **Principios de planejamento dos sistemas de transporte urbano**. Guanabara Dois, 1979. Disponível em: <http://books.google.com.br/books?id=Wn3WZwEACAAJ>. Acesso em: 20 mar. 2015.

MARTIN, B. V.; MANHEIM, M. L. A research program for comparison of traffic assignment techniques. **Highway Research Record**, n. 88, 1965.

PAPACOSTAS, C. S.; PREVEDOUROS, P. D. **Transportation engineering and planning**. [S.l.: s.n.], 1993.

SAADATSERESHT, M.; MANSOURIAN, A.; TALEAI, M. Evacuation planning using multiobjective evolutionary optimization approach. **European Journal of Operational Research**, Elsevier, v. 198, n. 1, p. 305–314, 2009.

UOL. **Tufão Haiyan matou ao menos 10 mil em província filipina, diz autoridade**. 2013. Disponível em: <http://noticias.uol.com.br/internacional/ultimas-noticias/2013/11/10/tufao-hayan-matou-ao-menos-10-mil-em-provincia-filipina-diz-autoridade.htm>. Acesso em: 01 dez. 2014.

VIRGULA. **Com 830 mil mortos, mais trágico terremoto completa 454 anos**. 2010. Disponível em: <http://virgula.uol.com.br/legado/com-830-mil-mortos-mais-tragico-terremoto-completa-454-anos/>. Acesso em: 01 dez. 2014.

APÊNDICES

A IMPLEMENTAÇÃO DO ALGORITMO NA LINGUAGEM C

```
/* Programa para determinar as rotas de fuga para uma população
* em áreas de risco, com várias origens e vários destinos.
*
* autores: Jônatas Trabuco Belotti e Sheila Morais de Almeida
* data: 14/10/2015
* versão: 2.0
*
* Implementa o método de alocação de fluxo em redes para evacuação
* de populações proposto por Belotti e Almeida em Trabalho de
* Conclusão de Curso apresentado a UTFPR Campus Ponta Grossa, Paraná.
* Explicações mais detalahadas sobre o método podem ser encontradas
* no documento impresso.
*
*/

#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include <limits.h>

//Struct para a fila usada na busca em largura
struct fila{
    int val;
    struct fila *prox;
};

int numVertices, numArestas, numAreasRisco, numAbrigos, inicio, destino,
removeu = 0, popAreasRisco = 0, popAbrigos = 0;
int **grafo, **redeResidual, *pai, **fluxo;

//Função responsável por inserir elementos na fila
struct fila * insereFila(struct fila *p, struct fila **fim, int v) {
    if(p == NULL){
        p = (struct fila *)malloc(sizeof(struct fila));
        if (p == NULL) {
            printf("Elemento não empilhado por falta de memória!\n");
            exit(0);
```

```
        }
        p->val = v;
        p->prox = NULL;
        *fim = p;
        return p;
    }else{
        (*fim)->prox = (struct fila *)malloc(sizeof(struct fila));
        (*fim)->prox->val = v;
        (*fim)->prox->prox = NULL;
        (*fim) = (*fim)->prox;
        return p;
    }
}

//Função responsável por remover determinado elemento da fila
struct fila * removeFila(struct fila *p){
    if(p == NULL){
        return NULL;
    }
    struct fila *aux = p;
    p = p->prox;
    free(aux);
    return p;
}

//Função responsável por remover todos os elementos da fila
struct fila * limpaFila(struct fila *fila, struct fila **fim){
    if(fila == NULL){
        *fim = NULL;
        return NULL;
    }else{
        struct fila *p = NULL;
        while(fila != NULL){
            p = fila;
            fila = fila->prox;
            free(p);
        }
        *fim = NULL;
        return fila;
    }
```

```
}

//Função responsável por inicar o grafo e a rede residual com os valores
//referentes a vazio
void iniciaValores(){
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            grafo[i][j] = -1;
            redeResidual[i][j] = 0;
        }
    }
}

/* Função responsável por ler todos os dados.
* Primeiramente são lidos a quantidade de vértices e arestas do grafo
* Depois são lidas as arestas do grafo
* Depois são lidas a quantidade de áreas de risco e abrigos
* Depois são lidas as áreas de risco
* Depois são lidos os abrigos
*/
void lerDados(){
    scanf("%d", &numVertices);
    scanf("%d", &numArestas);

    //Acrescentando a superorigem e o superdestino no grafo
    numVertices += 3;
    inicio = numVertices - 2;
    destino = numVertices - 1;

    //Alocando dinamicamento o grafo, a rede residual e a estrutura
    //responsvel por guardar as rotas determinadas
    grafo = (int **)malloc(numVertices * sizeof(int *));
    fluxo = (int **)malloc(numVertices * sizeof(int *));
    redeResidual = (int **)malloc(numVertices * sizeof(int *));
    pai = (int *)malloc(numVertices * sizeof(int));

    //Verificando se as estruturas realmente foram alocadas
    if(grafo == NULL){
        printf("Grafo não alocado!\n");
        exit(0);
```

```
}
if(fluxo == NULL){
    printf("Fluxo não alocado!\n");
    exit(0);
}
if(redeResidual == NULL){
    printf("redeResidual não alocado!\n");
    free(grafo); exit(0);
}
if(pai == NULL){
    printf("Pais não alocado!\n");
    free(grafo);
    free(redeResidual);
    exit(0);
}

//Segunda parte da alocação dinâmica das matrizes
for (int i = 0; i < numVertices; i++) {
    grafo[i] = (int *)malloc(numVertices * sizeof(int));
    fluxo[i] = (int *)malloc(numVertices * sizeof(int));
    redeResidual[i] = (int *)malloc(numVertices * sizeof(int));

    //Verificando se as estruturas realmente foram alocadas
    if(grafo[i] == NULL){
        printf("Grafo %d não alocado!\n", i);
        exit(0);
    }
    if(fluxo[i] == NULL){
        printf("Fluxo %d não alocado!\n", i);
        exit(0);
    }
    if(redeResidual[i] == NULL){
        printf("redeResidual %d não alocado!\n", i);
        exit(0);
    }
}

//Iniciando o grafo e a rede residual
iniciaValores();
```

```
    //Lendo as arestas do grafo juntamente com suas capacidades
    int v1, v2, c;
    for (int i = 0; i < numArestas; i++){
        scanf("%d %d %d", &v1, &v2, &c);
        grafo[v1][v2] = c;
        if(grafo[v2][v1] == -1){
            grafo[v2][v1] = c;
        }
    }

    //lendo as áreas de risco juntamente com a quantidade de UTs
    int a, ca;
    scanf("%d %d", &numAreasRisco, &numAbrigos);
    for (int i = 0; i < numAreasRisco; i++) {
        scanf("%d %d", &a, &ca);

        // Criando a aresta entre a superorigem e a área de risco com
        // capacidade igual a quantidade de UTs presentes na área de risco
        grafo[inicio][a] = ca;
        popAreasRisco += ca;
        grafo[a][a] = ca;
    }

    //lendo os abrigos juntamente com suas capacidades
    for (int i = 0; i < numAbrigos; i++) {
        scanf("%d %d", &a, &ca);

        //Criando a aresta entre o abrigo e o superdestino com capacidade
        // igual a capacidade do abrigo
        grafo[a][destino] = ca*(-1);
        popAbrigos += ca*(-1);
        grafo[a][a] = ca;
    }
}

//Função responsável por desalocar todas as estruturas que foram
// alocadas dinâmicamente
void limparDados(){
    for (int i = 0; i < numVertices; i++) {
        free(grafo[i]);
```

```
        free(fluxo[i]);
        free(redeResidual[i]);
    }
    free(grafo);
    free(fluxo);
    free(redeResidual);
    free(pai);
}


//Função responsável por realizar a busca em largura no grafo
int bfs(){
    // Cria um vetor para marcar quais vértices já foram visitados
    int *visited, n;
    visited = (int *)malloc(numVertices * sizeof(int));
    if(visited == NULL){printf("Falta de memoria!\n");exit(0);}
    for(int i = 0; i < numVertices; i++){
        visited[i] = 0;
    }

    // Cria a fila para executar a busca em largura
    struct fila *q = NULL, *fim = NULL;
    q = insereFila(q, &fim, inicio);
    visited[inicio] = 1;
    pai[inicio] = -1;

    // Inicia a busca em largura
    while (q != NULL){
        int u = q->val;
        q = removeFila(q);

        for (int v=0; (v<numVertices); v++){
            if (visited[v]==0 && redeResidual[u][v] > 0){
                q = insereFila(q, &fim, v);
                pai[v] = u;
                visited[v] = 1;
                if(v == destino){
                    n = visited[destino];
                    free(visited);
                    q = limpaFila(q, &fim);
```

```
                    return (n == 1);
                }
            }
        }
    }

    // Retorna verdadeira caso encontre um novo caminho da superorigem
    // até o superdestino
    // Retorna falso, caso contrário
    n = visited[destino];
    free(visited);
    q = limpaFila(q, &fim);
    return (n == 1);
}


/* Função que implementa o método de Ford-Fulkerson responsável por
 * determinar o valor do fluxo máximo.
 * Como as rotas são guardas na matriz fluxo é possível saber quais
 * as rotaas usadas para atingir o fluxo máximo.
*/
int fordFulkerson(){
    int u, v;

    //Iniciando redeResidual
    for (u = 0; u < numVertices; u++){
        for (v = 0; v < numVertices; v++){
            if(grafo[u][v] != -1){
                redeResidual[u][v] = grafo[u][v];
            }else{
                redeResidual[u][v] = 0;
            }

             fluxo[u][v] = 0;
         }
    }

    int max_flow = 0;  //iniciando o fluxo máximo com 0

    //Enquando existir um novo caminho entre a superorigem
```

```
    // e o superdestino
    while(bfs()){
        // Procura a aresta de menor capacidade no caminho
        // enconntrado, o valor do fluxo deste caminho é
        // igual a capacidade dessa aresta
        int path_flow = INT_MAX;
        for (v=destino; v!=inicio; v=pai[v]){
            u = pai[v];
            if(path_flow > redeResidual[u][v]){
                path_flow = redeResidual[u][v];
            }
        }

        // Atualiza  rede residual com o novo fluxo encontrado
        for (v=destino; v != inicio; v = pai[v]){
            u = pai[v];
            redeResidual[u][v] -= path_flow;
            //Impede que tenha retorno de fluxo
            redeResidual[v][u] = 0;

            // guarda a rota na matriz fluxo para que ela possa
            // ser usada futuramente
            fluxo[u][v] += path_flow;
        }

        // Adiciona o fluxo encontrado no fluxo máximo
        max_flow += path_flow;
        // imprimeCaminho(destino);
    }

    // retorna o valor do fluxo máximo encontrado
    return max_flow;
}

/* Função responsável por verificar se as rotas atendem as
 * restrições estabelecidas.
 * Verifica em todos os vértices e retira a aresta de menor fluxo
*/
int verificaFluxo(){
    int r = 1;
```

```
// Impede que arestas incidentes na superorigem ou superdestino
// sejam retiradas
int limite = numVertices - 2;

//Verifica para todos os vértices quais não atendem as restrições
for (int i = 0; i < limite; i++) {
    int quantSai = 0, menSai = i, quatEnt = 0, menEnt = i, menor;
    fluxo[i][i] = INT_MAX;

    //As rotas não são verificadas para as áres de risco ou abrigos
    // if (grafo[i][i] != -10){
        // Calcula quantoas fluxos entram no vértice e quantos
        // fluxos saem do vértice
        for(int j = 0; j < limite; j++) {
            // As restrições apenas são verificadas para vértices
            // que não são abrigos ou áres de risco
            // if(grafo[j][j] != -10){
                if((i != j) && (fluxo[i][j] > 0)){
                    quantSai++;
                    // Guarda qual aresta com menor fluxo que sai
                    // do vértice
                    if(fluxo[i][j] < fluxo[i][menSai]){
                        menSai = j;
                    }
                }
                if((i != j) && (fluxo[j][i] > 0)){
                    quatEnt++;
                    // Guarda qual aresta com menor fluxo que entra
                    // no vértice
                    if(fluxo[j][i] < fluxo[menEnt][i]){
                        menEnt = j;
                    }
                }
            // }
        }

        // Verifica se mais de um fluxo entra no vértice e mais de
        // um fluxo sai do vértice
        if(quatEnt > 1 && quantSai > 1){
            removeu++;
```

```
                r = 0;

                // Remove aresta com menor fluxo do grafo
                if(fluxo[i][menSai] < fluxo[menEnt][i]){
                    // printf("Removeu %d->%d\n", i,menSai);
                    grafo[i][menSai] = -1;
                    grafo[menSai][i] = -1;
                    fluxo[i][menSai] = 0;
                    fluxo[menSai][i] = 0;
                }else{
                    // printf("Removeu %d->%d\n", menEnt, i);
                    grafo[menEnt][i] = -1;
                    grafo[i][menEnt] = -1;
                    fluxo[menEnt][i] = 0;
                    fluxo[i][menEnt] = 0;
                }
            }
        // }
        fluxo[i][i] = 0;
    }

    //Retorna se as rotas estavam de acordo com as restrições
    // 1- Sim
    // 0 -Não
    return r;
}

// Função responsável por escoar o fluxo e atualizar os valores das
// áreas de risco e abrigos
void escoaFluxo(){
    int limite = numVertices-2;

    for(int i = 0; i < limite; i++){
        //Verificando áreas de risco
        if(fluxo[inicio][i] > 0){
            grafo[i][i] -= fluxo[inicio][i];
            grafo[inicio][i] = grafo[i][i];
        }

        // Verificando abrigos
```

```
        if(fluxo[i][destino] > 0){
            grafo[i][i] += fluxo[i][destino];
            grafo[i][destino] = grafo[i][i]*-1;
        }
    }
}

//Função responsável por imprimir as rotas de fuga
void imprimeFluxo(){
  int limite = numVertices - 2;
    for(int i = 0; i < limite; i++){
        for(int j = 0; j < limite; j++){
            if(fluxo[i][j] > 0){
                printf("%d -> %d (%d)\n",i,j,fluxo[i][j]);
            }
        }
    }
}

// Função principal
int main(){
    int f = 1, t = 1, i =1, pe = 0;
    float p = 100.0;
    setlocale(LC_ALL, "Portuguese");

    // Lê os dados de entrada
    lerDados();

    // Verificando se será possivel evacuar toda população, caso não
    // apresenta qual a porcentagem da população será evacuada
    if(popAreasRisco > popAbrigos){
        p = ((double)100/(double)popAreasRisco)*(double)popAbrigos;
        printf("Atenção!\nOs abrigos não tem capacidade para atender
               a todos, apenas %d das %d UTs (%.2f%%) serão salvas!\n\n",
               popAbrigos, popAreasRisco, p);
    }

    // Enquanto existir fluxo das áreas de risco para os abrigos
    while(f > 0){
        do{
```

```
            f = fordFulkerson();
        }while(!verificaFluxo());

        if(f > 0){
            //Escoando fluxo e atualizando valores
            escoaFluxo();

            // Imprimindo as rotas da leva
            printf("--Rotas do %d° turno----------------------\n",i);
            imprimeFluxo();

            //Imprimindo o fluxo total da leva
            printf("Fluxo: %d\n\n", f);

            pe += f;
            t++;
            i++;
        }
    }

    // Verifica se alguma população foi evacuada
    if(pe > 0){
        //Imprimendo o tempo total da evacuação
        printf("População evacuada: %d de %d UTs\n", pe, popAreasRisco);
        printf("O tempo total para evacuar %.2f%% da população é: %d
                unidades de tempo.\n\n",p,t);
    }else{
        printf("Não é posspivel evacuar ninguém!\n");
    }


    // Desalocando as estruturas dinâmicas
    limparDados();
    return 0;
}
```