An Optimized Breadth-First Search Algorithm for Routing in Optical Access Networks

G. Lopes, and D. Hoffman

***Abstract*—This work consists in the application of an optimized breadth-first search (BFS) algorithm to select a couple of link-and-node-disjoint shortest-path between the two most remote users within an optical access network. Our results showed that while the average execution time of the original BFS algorithm was 12.23 seconds that of the optimized BFS was 10.80 seconds, which means a reduction of 11.69 percent of the average computing time. In future works we intend to apply the modified BFS algorithm to a generic optical network and evaluate its performance for different mesh size, node concentration and random link arrangement.**

***Index Terms*—Breadth-first search, Network optimization, Optical networks, Routing, Shortest-path search algorithm.**

I. Introdução

A DEMANDA crescente de uso de dados, fomentada pelo maior acesso aos dispositivos móveis, pela adesão maciça às redes sociais e pelo uso em larga escala de aplicações em tempo real, como download de filmes e jogos em rede, resultou em uma maior abrangência e complexidade das redes ópticas atuais caracterizadas por uma grande carga de tráfego [1]. Neste contexto, algoritmos computacionais têm sido a ferramenta mais eficiente tanto no projeto quanto na gestão de tráfego de dados das redes, realizando funções como a escolha da rota (*routing*), a definição do melhor ou menor caminho e do caminho de proteção, a escolha do melhor canal de comunicação e ainda ações de contingência e contenção às falhas [1] [2]. Outras iniciativas de otimização do emprego dos recursos disponíveis nas redes ópticas envolvem o aprimoramento da qualidade de serviço em ações de roteamento através do emprego de algoritmos genéticos [3], a definição do nível ótimo da potência lançada na fibra para combater as não-linearidades, a dispersão do sinal e maximizar o número de usuários simultâneos [4] e também a virtualização das funções da rede de modo a potencializar o compartilhamento dos recursos da camada física [5].

Entre os algoritmos de busca mais populares estão o algoritmo de Dijkstra [6] e o algoritmo de busca em largura (*Breadth-First Search* – BFS) [7]. O algoritmo Dijkstra é considerado um algoritmo "*greedy*", ou seja, escolhe a melhor opção a cada passo, sem considerar passos futuros em seu processo de busca do menor caminho entre dois nós da rede, sendo a métrica mais usual a distância geográfica [1]. Por sua vez, o algoritmo BFS usa como métrica o número de enlaces (*hops*) para retornar a melhor solução tomando como base as informações gerais da rede em questão, ou seja, dentre todos os caminhos possíveis busca primeiramente aqueles constituídos de um único *hop*, depois aqueles com dois *hops* e assim sucessivamente, fornecendo ao final o caminho com menor número de *hops* ou a classificação destes em ordem crescente do número de *hops* [1].

O algoritmo BFS, também conhecido como *busca em amplitude* [8], é um algoritmo fundamental da teoria dos grafos, comumente utilizado para realizar a busca dos caminhos mais curtos em grafos não ponderados, dados os nós de origem e destino [7] [9]. Recentemente este algoritmo vem sendo utilizado para solucionar problemas computacionais desafiadores em redes Ponto-a-Ponto [10] [11] [12], onde é utilizado para a localização de pontos para a formação da rede [12] e também para escolher a melhor rota de compartilhamento de mensagens ou arquivos entre os nós da rede [11]. O uso do BFS também é frequente em redes elétricas inteligentes com o objetivo de minimizar o número de saltos entre os nós da rede em malha e obter o posicionamento ideal de concentradores GPRS (*General Packet Radio Service*) [13] [14]. O serviço GPRS é uma extensão do GSM (*Global System for Mobile Communications*), com a diferença que a transferência dos dados é através realizada através da comutação de pacotes. O GSM é o padrão celular digital pan-Europeu publicado pelo ETSI (*European Telecommunications Standards Institute* para transmissão de dados em telefonia móvel [15] [16]. Da mesma forma, pesquisas recentes tem explorado o uso do BFS em processadores *multicore* [8] [17] [18] [19]. Estes processadores possuem múltiplos núcleos de processamento para realizar diversas tarefas simultâneas (paralelas) e assim obter desempenho superior quando comparados aos processadores de um único núcleo. Neste caso, o BFS é utilizado em aplicações paralelas, ou seja, atua entre os diversos núcleos do processador para reduzir o tempo de busca. Além disso, o BFS é empregado para reduzir o tempo de acesso em processamentos computacionais que empregam memória distribuída [20] .

O algoritmo BFS também pode ser utilizado em jogos digitais, como o *Maze Runner* [21] e o *Bomberman* [22]. Ambos são jogos baseados em labirintos em que o objetivo é vencer o adversário através do emprego do menor caminho para a saída.

Algoritmos de busca apresentam aplicação crescente para realizar o roteamento em redes ópticas, ou seja, para encontrar o menor caminho (*shortest-path*) entre usuários destas redes [2], sendo que o menor caminho será a soma das métricas dos enlaces que compõem o caminho. As métricas são as mais variadas possíveis, sendo bastante comuns a distância geográfica, o número de *hops* e a probabilidade de disponibil-

G. L. Andrade, Universidade Federal do Pampa, Alegrete, Rio Grande do Sul, Brasil, gabie.lop.s@gmail.com.

D. H. Thomas, Universidade Federal do Pampa, Alegrete, Rio Grande do Sul, Brasil, djeissonthomas@unipampa.edu.br.

idade. A métrica da distância geográfica, medida usualmente em quilômetros (km), busca o caminho mais curto, independentemente do número de enlaces (*links*) e nós (*nodes*) deste caminho. Já a métrica do menor número de *hops* ou saltos considera cada enlace da rede como uma unidade ou *hop* e, desta forma, o menor caminho é aquele composto pelo menor número possível de *hops*. Por sua vez, na métrica da probabilidade de disponibilidade, a disponibilidade do caminho é igual ao produto das disponibilidades dos enlaces do caminho. Logo, se tomarmos como meta o negativo do logaritmo da disponibilidade, uma maior disponibilidade corresponderá a uma menor métrica para o enlace. Desta forma, ao executar um algoritmo de busca para esta métrica, o menor caminho será o caminho com maior disponibilidade [1].

No contexto do projeto de redes ópticas, um menor número de *hops* significa que a transmissão se dará através de um menor número de enlaces e nós, ou seja, garante uma conexão mais direta, de menor custo e com menor probabilidade de falhas quando comparada com aquela que seria obtida através da menor distância geográfica [1]. Logo, optou-se pelo algoritmo BFS para realizar a função de roteamento, ou seja, encontrar um par de menores caminhos totalmente distintos (*link-and-node-disjoint*) entre os dois usuários mais afastados de uma rede óptica de acesso [1] [23] [24]. Tomando como base o resultado obtido pelo BFS original, foram propostas algumas simplificações com o intuito de disponibilizar uma versão mais eficiente do código, com reduzido número de passos, capaz de alcançar o objetivo proposto com menor custo computacional.

O restante deste trabalho está estruturado da seguinte forma: a Seção II apresenta a descrição do problema; a Seção III apresenta o passo-a-passo da operação do algoritmo BFS original e o código correspondente; a Seção IV apresenta o código do algoritmo BFS otimizado, com base nas alterações propostas; a Seção V descreve o ambiente de execução utilizado para os testes; a Seção VI apresenta os resultados obtidos e a discussão sobre a eficiência do novo código na resolução do problema alvo e a Seção VII apresenta a conclusão e as perspectivas de trabalhos futuros.

II. Problema Alvo

A rede alvo deste trabalho é uma rede de acesso estruturada em malha, a qual foi projetada para atender 203 clientes do bairro Nova Brasília da cidade de Alegrete, no estado do Rio Grande do Sul, Brasil. A Fig. 1 apresenta a rede alvo, cujo mapeamento da área foi realizado utilizando a ferramenta Google Earth Pro em sua versão 7.8 [25], ou seja, a localização dos nós nesta figura corresponde à localização geográfica das residências dos potenciais clientes da rede óptica proposta, de acordo com as informações públicas fornecidas pela ferramenta.

No contexto de redes ópticas, deseja-se encontrar um par de menores caminhos totalmente distintos (*link-and-node-disjoint*) [1] entre os dois usuários mais afastados dentre os 203 que compõe a rede óptica de acesso em malha. Dito de outra forma e considerando-se a opção pelo algoritmo BFS, deve-se encontrar dois caminhos com o menor número de *hops*

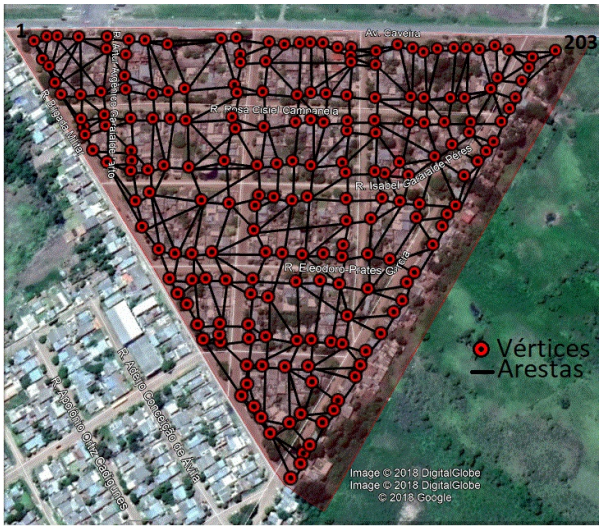

Fig. 1. Rede de acesso em malha para atender 203 clientes do bairro Nova Brasília, da cidade de Alegrete, Rio Grande do Sul, Brasil.

possíveis, sendo que nenhum enlace ou nó pode se repetir ao longo de ambos os caminhos, exceto os nós de origem e destino. Os nós de origem e de destino foram identificados como 1 e 203, respectivamente, e estão dispostos nas duas extremidades da Avenida Caverá, como pode ser observado na Fig. 1.

Além disto, é importante dispôr de um segundo menor caminho como forma de proteção, ou seja, para pronto atendimento (sem atrasos adicionais) de qualquer solicitação de demanda de dados entre dois pares de nós da rede óptica caso o primeiro menor caminho estiver indisponível, seja esta indisponibilidade física (rompimento da fibra) ou lógica (congestionamento de tráfego) [1].

III. Algoritmo de Busca em Largura

A rede em estudo (Fig. 1) será tratada como um grafo não orientado, sob o qual será aplicado o algoritmo de busca BFS, de forma a encontrar o par de caminhos mínimos e distintos. Como não está entre os objetivos calcular a distância geográfica, o peso de cada aresta ou link será normalizado para a unidade, ou seja, é considerado igual a 1 (um).

No BFS, a rede é considerada como um grafo $G = (V, E)$ não direcionado ou não orientado [26], onde V representa um conjunto não vazio de vértices pertencentes ao grafo G e E o conjunto de arestas que ligam os vértices entre si [27]. Considerando o grafo G e um vértice de origem s pertencente a G, o algoritmo BFS explora sistematicamente as arestas de G até descobrir cada vértice acessível a partir do vértice de origem s. O BFS visita s, depois visita todos os vértices vizinhos de s, depois todos os vértices que estão à distância 2 de s, depois todos os que estão à distância 3 de s, e assim por diante. O BFS numera os vértices em sequência, na ordem crescente em que são descobertos [7].

O algoritmo BFS produz uma árvore chamada de primeiro na extensão [7], na qual tem-se o vértice de origem s como raiz. Sempre que um vértice v é descoberto no curso da

varredura dos vértices adjacentes de um vértice u já descoberto, o vértice v e a aresta (u, v), que representa uma ligação entre os vértices u e v, são adicionados à árvore. Logo, diz-se que u é predecessor ou pai de v na árvore, e consequentemente v é filho ou descendente de u [26]. Cada vértice é descoberto no máximo uma vez, logo ele tem no máximo um pai. Porém, um vértice pode ter mais de um filho [7]. A partir da árvore de primeiro na extensão pode-se obter o menor caminho de s a qualquer vértice.

A Fig. 2 ilustra o funcionamento do algoritmo BFS sobre um grafo não direcionado com 6 vértices e 6 arestas. Para programar o algoritmo BFS utiliza-se uma fila Q, a qual controla a ordem em que os vértices são descobertos na busca [26]: os vértices ainda não visitados são representados em "branco", aqueles visitados mas que possuem vizinhos ainda não visitados, em "cinza", e os vértices com seus vizinhos já visitados são representados em "preto" [7]. Inicialmente todos os vértices são "brancos" e a distância é dada como infinita, ou seja, a distância que cada vértice possui em relação à raiz s ainda não foi calculada. No começo de cada iteração do algoritmo, a fila Q contém os vértices que já foram visitados, mas possuem vizinhos ainda não visitados. Por conseguinte, na primeira iteração a fila Q contém apenas o vértice s identificado com o número "0" e pintado de "cinza" (Fig. 2(a)). No passo seguinte, os vértices vizinhos de s, r e v, são visitados, pintados de "cinza", inseridos na fila Q e numerados como "1" e, consequentemente, s é pintado de "preto" e retirado da fila Q (Fig. 2(b)). O próximo vértice a ter seus vizinhos visitados sempre será o primeiro da fila Q, ou seja, o vértice v. Os vizinhos de v, vértices t e w, são visitados, pintados de "cinza", inseridos na fila Q e numerados como "2" e v é pintado de "preto" e retirado da fila Q, (Fig. 2(c)).

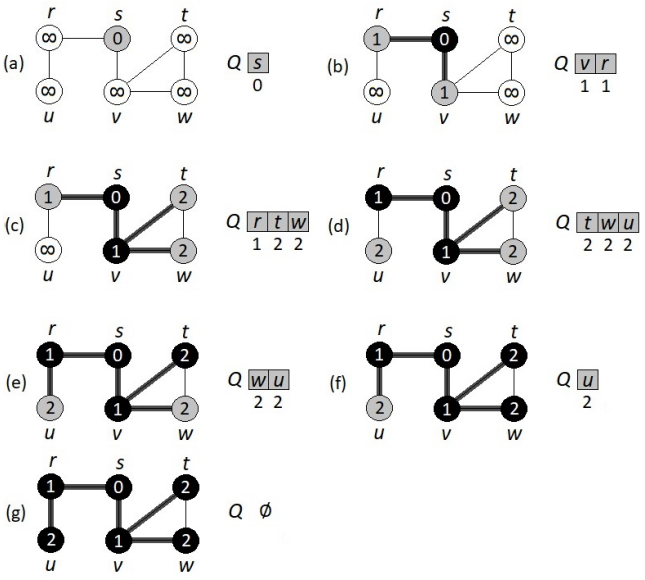

Fig. 2. Passo-a-passo da operação do BFS original sobre um grafo não-orientado [7].

Em seguida, o vértice vizinho de r, u, é visitado, pintado de "cinza", inserido na fila Q e numerado como "2" e r é pintado de "preto" e retirado da fila Q (Fig. 2(d)). Caso todos os vizinhos de um vértice já tenham sido visitados, como é o caso dos vértices t, w e u, o vértice é diretamente pintado de "preto" e retirado da fila Q, sem incremento de numeração. Esse procedimento do BFS é repetido até que todos os vértices tenham sido pintados de "preto" e a fila Q esteja vazia, finalizando a execução do BFS (Fig. 2(e-g)) [7].

Para implementar o BFS, representa-se o grafo de forma que possa ser processado computacionalmente. Existem diversas formas alternativas para a representação computacional de grafos. As mais utilizadas são as estruturas de dados conhecidas como *listas de adjacências* e *matriz de adjacências* [7] [27]. Optou-se pela matriz de adjacências para realizar a representação do grafo pois o algoritmo proposto será implementado na linguagem de programação *Matlab*, a qual é muito intuitiva na criação e manipulação de matrizes. Na matriz de adjacências de um grafo $G = (V, E)$, se supõem que os vértices são numerados como 1, 2, ... $|V|$ de alguma maneira arbitrária. Logo, a representação da matriz de adjacências de um grafo G consiste em uma matriz $A_{|V|\times|V|}$, onde a dimensão da matriz quadrada A será igual ao número total de vértices $|V|$. Caso haja uma aresta ligando um vértice i a um vértice j, o peso da aresta será igual a 1, ou seja, $A_{ij} = 1$. Caso contrário, o peso da aresta será igual à 0, sendo $A_{ij} = 0$ [7]. Nas redes ópticas, os vértices representam os nós e as arestas as ligações entre cada par de nós que compõe a rede.

Além da matriz de adjacências A do grafo G, que será dada como entrada ao algoritmo BFS, é considerada a inserção e remoção dos vértices na fila Q [26]. Para representar a cor de um vértice utilizou-se um vetor cores, cujos elementos podem receber a atribuição "branco" (ainda não visitado), "cinza" (visitado com seus vizinhos ainda não visitados) ou "preto" (visitado com seus vizinhos já visitados). Para armazenar os dados da árvore de busca em largura, utilizou-se um vetor pa, o qual armazenará o predecessor ou pai de cada vértice pertencente ao menor caminho. Para armazenar a distância dos vértices em relação à origem s, se utiliza um vetor $dist$. O algoritmo 1 apresenta um pseudocódigo que ilustra o funcionamento do BFS original [7].

Inicialmente o algoritmo BFS recebe como entrada a matriz de adjacências Adj do grafo G e o vértice de origem s. Para cada vértice u de G, exceto o vértice de origem s (linha 2), o vetor de $cores$ será igual a "branco" (linha 3), ou seja os vértices ainda não foram visitados; o vetor de pais pa será (linha 4), visto que ainda não foi definido o predecessor dos vértices; e o vetor de distâncias $dist$ será igual a infinito (linha 5), visto que a distância em relação à origem s ainda não foi medida [7]. Logo após, define-se para o vértice s $cores$ igual à "cinza" (linha 7), indicando que s foi visitado mas seus vizinhos ainda não, pa igual a nulo ou seja, s não tem pai (linha 8) e $dist$ igual a "0", ou seja, s é o vértice de origem (linha 9). Até este passo, a fila Q está vazia (linha 10), então insere-se s no fim da fila (linha 11).

Enquanto a fila Q não estiver vazia (linha 12), retira-se do inicio da fila Q o primeiro elemento v (linha 13). Cada vértice w adjacente à v que estiver "branco" terá como pai v (linha 16). O vértice w será inserido no final da fila (linha 17) e pintado de "cinza", ou seja, $cores$ igual à "cinza" (linha 18). Após a fila Q ficar vazia, o algoritmo BFS retorna o vetor dos

Algorithm 1 Algoritmo de Busca em Largura

```
1: função BFS(Adj, s)
2:     para cada vértice u em V[G] − s ← até faça
3:         cores[u] ← branco
4:         pa[u] ← NULL
5:         dist[u] ← ∞
6:     fim para
7:     cores[s] ← cinza
8:     pa[s] ← 0
9:     dist[s] ← 0
10:    Q ← ∅
11:    ENQUEUE(Q, s)
12:    enquanto Q ≠ ∅ faça
13:        v ← DEQUEUE(Q)
14:        para cada vértice w em Adj[u] ← até faça
15:            se cores[w] = branco então
16:                pa[w] ← v
17:                ENQUEUE(Q, w)
18:                cores[w] ← cinza
19:            fim se
20:        fim para
21:        cores[v] ← preto
22:    fim enquanto
23:    devolve pa
24: fim função
```

pais pa, a partir do qual pode-se recuperar o menor caminho da origem s até qualquer vértice de destino. O menor caminho (número de *hop*) será a distância $dist$ do vértice de origem s até o destino desejado y, acrescido de 1. Para remontar o caminho utilizou-se o vetor pa em ordem inversa, ou seja, contabilizando os predecessores de cada vértice que compõe o caminho desde o vértice de destino até o de origem. O algoritmo 2 apresenta um pseudocódigo que ilustra os passos necessários, o qual recebe como entrada o vértice de destino y e o vetor pa.

Algorithm 2 Função que Recupera o Caminho Mínimo

```
1: função RECUPERA(y, pa)
2:     i ← 1
3:     x[i] ← y
4:     v ← y
5:     i ← i + 1
6:     enquanto pa[v] ≠ 0 faça
7:         x ← pa[v]
8:         v ← x[i]
9:         i ← i + 1
10:    fim enquanto
11:    Caminho Mínimo ← fliplr(x)
12:    devolve Par de Caminhos Mínimos Distintos
13: fim função
```

No algoritmo 2, utilizou-se uma variável auxiliar i (linha 2) que é incrementada (linha 5) para percorrer todo o vetor de pais pa; um vetor x para armazenar o caminho, começando pelo vértice de destino y (linha 3), e uma variável v para armazenar temporariamente o próximo vértice do caminho (linha 4). Enquanto o vetor pa não for igual a 0 (zero) para um determinado vértice v (linha 6), ou seja, v não for o vértice de origem s, todos os elementos do vetor de pais pa são contabilizados. No início do laço de repetição, o vetor $x[i]$ recebe o pai do vértice de destino (linha 7), o novo vértice v será o pai $pa[v]$ (linha 8) e a variável i é incrementada (linha 9).

No final do algoritmo 2, utiliza-se uma função nativa do Matlab chamada "fliplr" (linha 11), a qual retorna o vetor x com todas suas posições invertidas, de modo que a última posição do vetor estará no lugar da primeira e assim por diante. Sendo assim, a função "fliplr" retornará o vetor x invertido, o qual indicará o caminho mínimo (linha 12).

Para encontrar o par de caminhos mínimos distintos, o BFS implementado (algoritmo 1) é empregado como uma função do programa principal, conforme pode ser observado no pseudocódigo do Algoritmo 3. A primeira chamada à função BFS (linha 2) resulta no primeiro caminho mínimo (linha 3). Logo após esta primeira chamada, a matriz de adjacências Adj é atualizada, eliminando-se todas as arestas que interligam os vértices do primeiro caminho (linha 4). A função BFS é então executada novamente (linha 5) retornando o segundo caminho mínimo (linha 6). Através desta sequência de passos, garante-se que o par de caminhos mínimos resultante também será totalmente distinto, ou seja, sem repetição de nenhum enlace ou *hop*, sendo coincidentes apenas os nós de origem e de destino. Como último passo, o BFS imprime na tela o resultado, ou seja, a sequência de *hops* que compõe cada um dos caminhos mínimos totalmente distintos (linha 7).

Algorithm 3 Algoritmo Principal

```
1: função BFS(Adj, s, y)
2:     BFS(Adj, s, y)
3:     Recupera(y, pa)
4:     Atualiza matriz de Adjacências
5:     BFS(AdjAtual, s, y)
6:     Recupera(y, paAtual)
7:     devolve Par de Caminhos Mínimos Distintos
8: fim função
```

A partir de técnicas de processamento de imagens [28] [29], este resultado pode ser representado sobre a imagem original (Fig. 1), que contém a disposição geográfica dos usuários da rede em estudo. Estas técnicas permitem não apenas identificar os nós e enlaces que compõe o caminho mínimo, mas também interligá-los desde o nó de origem até o destino, indicando o trajeto a ser percorrido pelo sinal de comunicação.

IV. Algoritmo de Busca em Largura Otimizado

Com base nos resultados obtidos pelo emprego do algoritmo BFS original, avalia-se a possibilidade de simplificá-lo e assim alcançar o objetivo proposto com menor custo computacional, ou seja, propor um BFS otimizado.

Da mesma forma que no pseudocódigo BFS original, considerou-se a inserção e remoção dos vértices na fila Q assim que forem sendo visitados [26] através da utilização de um vetor. Para representar se um vértice foi visitado, ou seja,

a cor do vértice, utilizou-se um vetor chamado $visit$. Caso o vértice em questão não tiver sido visitado (situação que corresponde ao status "branco" no algoritmo original) $visit$ será igual a 0 (zero); de forma complementar, se o vértice tiver sido visitado, $visit$ assume o valor 1 (um). A alteração simplifica o passo-a-passo do algoritmo original em que a cor do vértice deve ser alterada, de "branco" para "cinza" e posteriormente para "preto", na medida em que os vértices do caminho, e seus respectivos vértices vizinhos, forem sendo visitados (algoritmo 1).

Para armazenar os dados da árvore de busca em largura se utilizou um vetor pa. O vetor pa armazena o predecessor ou pai de cada vértice, permitindo recuperar o caminho mínimo desejado, desde o vértice de origem até o de destino.

No BFS original a distância é considerada como o número de saltos, a qual é calculada de acordo com a numeração dos vértices [7]. No pseudocódigo que proposto não foi considerada a numeração dos vértices, ao invés disto a distância entre os nós de origem e destino será calculada através da soma do número de enlaces ou *hops* que compõem o caminho, lembrando que foi considerado o peso de cada *hop* igual a 1 (um).

O algoritmo 4 apresenta um pseudocódigo com o funcionamento do algoritmo BFS otimizado, incluindo as modificações propostas. Ao observar o algoritmo 4, nota-se que enquanto a fila Q não estiver vazia (linha 10), retira-se do início da fila o primeiro elemento v (linha 11). Cada vértice w adjacente à v que não foi visitado, terá como pai v (linha 14). Testa-se se o vértice w adjacente à v é o vértice de destino y (linha 15) e, em caso positivo, esvazia-se a fila Q (linha 16) e finaliza-se a execução do BFS (linha 17). Essa condição de parada, que não existe no pseudocódigo original, pode conduzir a uma significativa redução do custo computacional, especialmente na situação em que o nó de destino não é o mais distante em relação ao nó de origem, evitando que o algoritmo continue executando de forma desnecessária. Caso a condição de parada não se verifique, o vértice w será inserido no final da fila e marcado como visitado (linhas 19 e 20). Após a fila Q ficar vazia, recupera-se o caminho da origem até o destino a partir do vetor dos pais pa, assim como no BFS original.

Também de forma análoga ao que foi feito para o pseudocódigo BFS original, o pseudocódigo do BFS otimizado foi empregado como uma função no algoritmo principal (Algoritmo 3). Por conseguinte, no final da execução retorna-se o par de caminhos mínimos totalmente distintos e utilizam-se as mesmas técnicas de processamento de imagem para representar a solução sobre o mapeamento original dos usuários da rede em estudo (Fig. 1).

Algorithm 4 Algoritmo de Busca em Largura Otimizado

```
1:  função BFS(Adj, s, y)
2:      para cada vértice u em V[G] − s ← até faça
3:          pa[u] ← NULL
4:          visit[u] ← ∞
5:      fim para
6:      pa[s] ← 0
7:      Q ← ∅
8:      ENQUEUE(Q, s)
9:      visit[s] ← 1
10:     enquanto Q ≠ ∅ faça
11:         v ← DEQUEUE(Q)
12:         para cada vértice w em Adj[u] ← até faça
13:             se visit[w] = 0 então
14:                 pa[w] ← v
15:                 se w = y então
16:                     Q ← ∅
17:                 senão
18:                     ENQUEUE(Q, w)
19:                     visit[w] ← 1
20:                 fim se
21:             fim se
22:         fim para
23:     fim enquanto
24:     devolve pa
25: fim função
```

V. Ambiente de Execução e Metodologia

Como ambiente de execução utilizou-se o *software* Matlab® [30] em sua versão R2017a, executado em um microcomputador com processador Intel® Core™ i5-7200U, com 2 núcleos físicos e 4 núcleos lógicos, com 2,5 GHz de frequência, memória RAM DDR4 de 4 GB e sistema operacional Windows 10 *Home Single Language* de 64 bits.

Para a rede de acesso em malha em estudo, foram realizadas dez execuções [31] dos pseudocódigos BFS original e otimizado com o intuito de comparar o desempenho de ambos e verificar o impacto das modificações propostas, ou seja, se as alterações oferecem economia em termos de custo computacional, viabilizando a proposição do novo código.

A Fig. 3 e a Fig. 4 apresentam os pares de caminhos traçados sobre o mapa da rede de acesso em questão, as quais foram geradas através da incorporação dos resultados da execução dos algoritmos BFS sobre o mapa original, utilizando-se para tanto técnicas de processamento de imagens [28].

VI. Análise dos Resultados

A análise da Fig. 3 e Fig. 4 indicam que o caminho em linha reta entre os vértices 1 e 203 possui 29 *hops*, intuitivamente considerado o menor caminho entre os vértices de origem e destino escolhidos para o estudo.

Por sua vez, a execução de ambos os algoritmos resultou no mesmos par de menores caminhos. O primeiro caminho encontrado com a execução de ambos os algoritmos BFS é composto de 18 *hops*, sendo eles: 1, 2, 4, 11, 24, 27, 28, 33, 32, 133, 124, 152, 150, 149, 146, 145, 144 e 203. Por sua vez, o segundo caminho é composto de 23 *hops*, sendo eles: 1, 3, 9, 19, 18, 17, 52, 49, 48, 106, 99, 169, 167, 157, 158, 159, 160, 161, 197, 198, 199, 200 e 203.

Como a rede óptica de acesso em questão é restrita a um bairro que resultou em um número limitado de 203 vértices e 435 arestas e, além disto, como o algoritmo otimizado elimina as arestas da sua matriz da adjacências uma vez que elas tenham sido utilizadas em um caminho, não foi possível encontrar mais caminhos distintos além do terceiro, mostrado

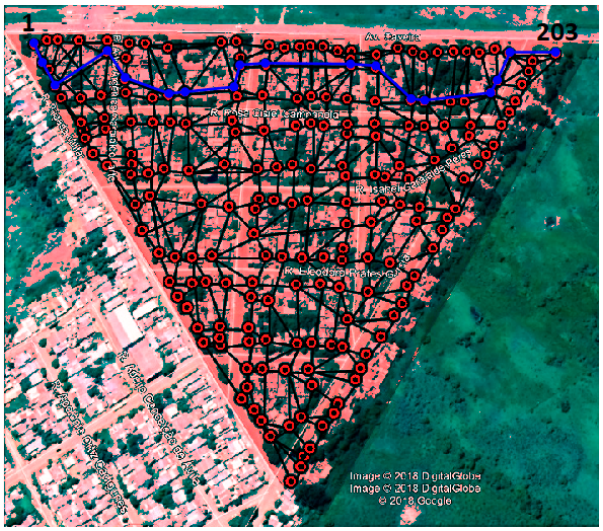

Fig. 3. Primeiro caminho mínimo entre os vértices de origem 1 e de destino 203.

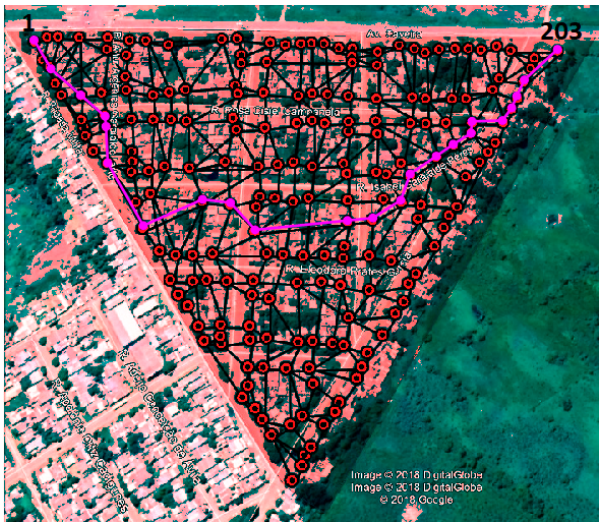

Fig. 4. Segundo caminho mínimos entre os vértices de origem 1 e de destino 203.

na Fig. 5. O terceiro caminho encontrado com a execução de ambos os algoritmos BFS é composto de 25 *hops*, sendo eles: 1, 6, 7, 12, 25, 26, 37, 36, 35, 34, 134, 120, 115, 114, 123, 151, 163, 162, 148, 141, 142, 143, 201, 202 e 203.

Os resultados obtidos indicam que ambos os algoritmos são capazes de obter a solução procurada, ou seja, um par de caminhos mínimos totalmente distintos. No entanto, há que se verificar a eficiência de ambos em obter o resultado. Para tanto, foi realizada uma análise estatística considerando 10 execuções [31]. A Tabela I apresenta a média dos tempos de 10 execuções (em segundos) e o desvio padrão desta média para encontrar um caminho, um par de caminhos totalmente distintos e três caminhos totalmente distintos pelos algoritmos BFS e BFS Otimizado. Como resultado, enquanto o algoritmo BFS original apresentou um tempo de execução médio de 5,14 segundos para encontrar um caminho, o BFS otimizado apresentou um tempo de execução de 1,07 segundos, com uma

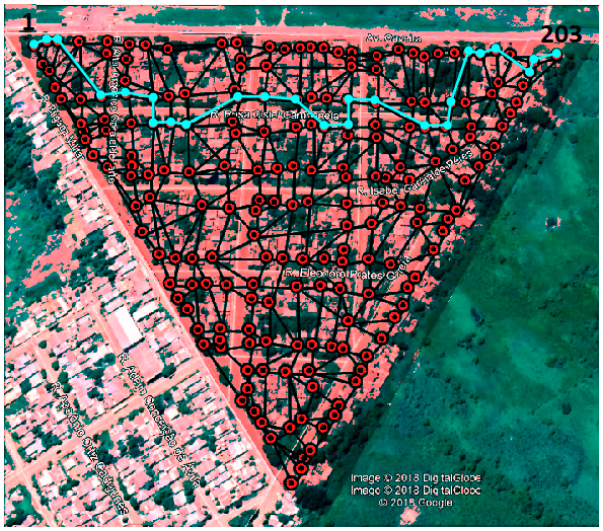

Fig. 5. Terceiro caminho mínimo entre os vértices de origem 1 e de destino 203.

redução de 79,18% no custo computacional. O algoritmo BFS original apresentou um tempo de execução médio de 12,23 segundos, o algoritmo BFS otimizado apresentou um tempo de execução médio de 10,80 segundos, com uma redução de 11,69%. E para encontrar três caminhos o BFS original apresentou um tempo de execução médio de 43,30 segundos para encontrar três caminhos, o algoritmo BFS otimizado apresentou um tempo médio de 30,38 segundos, com uma redução de 29,83%. O terceiro caminho serve apenas e exclusivamente para comprovar que o algoritmo otimizado é mais eficiente, independentemente do número de caminhos considerados.

TABELA I
CUSTO COMPUTACIONAL DOS ALGORITMOS BFS E BFS OTIMIZADO NA BUSCA POR CAMINHOS TOTALMENTE DISTINTOS ENTRE OS VÉRTICES DE ORIGEM 1 E DE DESTINO 203

Método		Caminhos Um	Totalmente Dois	Distintos Três
BFS	Tempo Médio (s)	5,14	12,23	43,30
	Desvio Padrão	2,03	5,97	8,37
BFS	Tempo Médio (s)	1,07	10,80	30,38
Otimizado	Desvio Padrão	0,04	2,70	8,39

A complexidade é idêntica pois os dois algoritmos, sendo $O(V + E)$ no pior caso. Em função disto, a comparação de desempenho entre os algoritmos passou a ser feita em função do tempo médio (após 10 execuções) necessário para a obtenção da solução, ou seja, para encontrar o par de caminhos mínimos. Neste sentido e dado que o tempo médio do algoritmo otimizado foi 11,69% menor, pode-se afirmar que o custo computacional (proporcional ao tempo dispendido para encontrar a solução) foi reduzido na mesma proporção.

Em função disto, a supressão de estruturas, como a que é obtida pela substituição do código de cores pelo vetor que indica se um determinado nó foi visitado ou não (vetor $visit$), e também o estabelecimento de uma condição de parada, representam, comprovadamente, uma redução signi-

ficativa do custo computacional necessário à obtenção da melhor solução pelo novo pseudocódigo proposto em relação ao pseudocódigo original. Logo, considera-se válida a proposição desta nova versão do pseudocódigo BFS enquanto ferramenta de aperfeiçoamento das ações de roteamento nas redes ópticas.

Por conseguinte e além das simplificações propostas, se destacou ainda o impacto do emprego da matriz de adjacências sobre a redução do tempo de execução, a qual foi fornecida como entrada ao algoritmo BFS. Desta forma, não foi preciso que o algoritmo realizasse cálculos para encontrar os vizinhos de cada vértice, sendo apenas necessário consultar esta informação na matriz de adjacências.

Por outro lado, convém mencionar que existe um custo inicial na formatação da matriz de adjacências, visto que esta foi elaborada antes da execução do BFS. Em função disto, o aumento da dimensão do problema traria desvantagens, visto que quanto maior for a dimensão do problema, maior será a dimensão da matriz de adjacências. Neste sentido, o teste de parada incorporado ao BFS otimizado para detectar quando o nó de destino foi alcançado é de grande importância, pois restringe o tempo computacional ao estritamente necessário, permitindo alcançar o custo computacional mínimo para cada par de nós origem-destino, especialmente na situação em que o nó de destino não é o mais distante em relação ao nó de origem dentre aqueles que compõem a rede.

VII. Conclusão

Neste trabalho propõe-se uma nova versão do pseudocódigo BFS enquanto ferramenta de aperfeiçoamento das ações de roteamento nas redes ópticas. O pseudocódigo do BFS otimizado foi empregado com sucesso para selecionar um par de menores caminhos totalmente distintos entre os dois usuários mais afastados de uma rede de acesso em malha composta por 203 clientes, considerando-se como métrica o número de enlaces ou *hops*. A rede em questão foi projetada para atender clientes do bairro Nova Brasília da cidade de Alegrete, estado do Rio Grande do Sul, Brasil.

Intuitivamente e tomando-se como métrica a menor distância geográfica, o menor caminho entre o nó de origem 1 e o nó de destino 203 seria a linha reta entre ambos, às margens da Avenida Caverá, no entanto esse caminho é formado por vinte e nove *hops*. Logo, os resultados obtidos mostraram que ambos os algoritmos BFS, original e otimizado, permitem encontrar um par de caminhos mínimos (menor número de *hops*) totalmente distintos, sendo o primeiro caminho composto de dezoito *hops* e o segundo de vinte e três *hops*. No contexto das redes ópticas, caminhos com número mínimo de *hops* representam a melhor solução porque viabilizam uma conexão mais direta entre os nós de origem e destino, ou seja, garante-se que a comunicação ocorra por uma rota composta por um menor número de enlaces e nós e, portanto, com menor probabilidade de falhas e de menor custo (em termos do número de equipamentos utilizados) para o atendimento da demanda.

No entanto, além de se obter a melhor solução, é necessário considerar a eficiência do processo de busca da mesma, o que é realizado através da comparação do custo computacional dispendido. Neste caso, a eficiência foi medida através da média de tempo de execução ao longo de dez repetições de ambos os algoritmos, necessárias à obtenção do resultado: enquanto o algoritmo BFS original apresentou tempo médio de execução de 12,23 segundos, o algoritmo otimizado alcançou o mesmo objetivo após um tempo médio de execução de 10,80 segundos. Logo, as simplificações propostas, como a que é obtida pela substituição do código de cores pelo vetor que indica se um determinado nó foi visitado ou não (vetor $visit$), garantiram um ganho de eficiência de 11,69% ao reduzir o tempo de execução, o que se reflete diretamente na proporcional diminuição do custo computacional.

O algoritmo BFS foi implementado na linguagem de programação Matlab, a qual se mostrou bastante apropriada ao uso da matriz de adjacências por se tratar de uma linguagem de programação afeita à manipulação e operação com matrizes de forma rápida e intuitiva. Além do custo computacional mínimo, a disponibilização do par de caminhos mínimos no menor tempo possível significa aprimorar a agilidade no atendimento de qualquer demanda de dados da rede óptica, ou seja, reduzir a latência da comunicação através desta rede.

Concluí-se que o BFS se mostrou eficiente para resolver o problema alvo, sendo bastante significativa a redução do custo computacional viabilizada pelo algoritmo otimizado. Disponibiliza-se portanto uma alternativa de otimização para o roteamento em redes ópticas, etapa cada vez mais importante no projeto destas redes, na medida em que a sua complexidade aumenta em função da crescente demanda dos serviços de dados atuais e futuros. Neste sentido, a técnica proposta contribui para a redução da latência da rede, parâmetro-chave na avaliação da qualidade destes serviços, especialmente com o advento do novo padrão de comunicação sem fio 5G e das aplicações em IoT (*Internet of Things* ou "Internet das Coisas"). Merecem destaque neste contexto as redes ópticas de acesso, as quais representam a interface com o usuário e são decisivas na qualidade da prestação do serviço de dados e também no aumento da cobertura destes serviços. A demonstração da aplicabilidade da nova versão do pseudocódigo BFS a estas redes reforça a significância da contribuição deste trabalho.

Em trabalhos futuros pretende-se utilizar o algoritmo BFS otimizado para avaliar o desempenho de uma rede óptica genérica, a qual é uma rede que apresenta diferentes tamanhos de malha, concentração de nós e disposição aleatória de links ou arestas.

Referências

[1] J. M. Simmons, *Optical network design and planning*. Springer, 2014.

[2] E. Israeli and R. K. Wood, "Shortest-path network interdiction," *Networks: An International Journal*, vol. 40, no. 2, pp. 97–111, 2002.

[3] R. Sun, S. Zhang, B. Zhang, and C. Zhang, "A novel multi-objective genetic algorithm for the qos based multicast routing and wavelength allocation problem in wdm network," in *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2017, pp. 286–291.

[4] V. Kachhatiya and S. Prince, "Performance analysis and optimization of wavelength routed (wr) and wavelength selected (ws) hybrid optical distributed network (odn) for next generation passive optical network stage 2 (ng-pon2)," *Optics & Laser Technology*, vol. 106, pp. 335–347, 2018.

[5] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, "Software defined optical networks (sdons): A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2738–2786, 2016.
[6] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Algoritmos: teoria e prática*. Rio de Janeiro: Elsevier, 2002, vol. 2.
[8] Y. Xia and V. K. Prasanna, "Topologically adaptive parallel breadth-first search on multicore processors," in *Proc. 21st Int'l. Conf. on Parallel and Distributed Computing Systems (PDCS'09)*. Citeseer, 2009.
[9] S. Beamer, K. Asanović, and D. Patterson, "Direction-optimizing breadth-first search," *Scientific Programming*, vol. 21, no. 3-4, pp. 137–148, 2013.
[10] G. do Carmo, J. Barbar, and F. Coelho, "Uma nova classificaçao para as arquiteturas de redes peer-to-peer," *Revista do IEEE América Latina*, 2006.
[11] N. A.-H. Al-Dmour, "Comparison of file sharing search algorithms over peer-to-peer networks," *International Journal of Engineering Research*, vol. 5, no. 3, pp. 203–206, 2016.
[12] M. V. Neves, T. Scheid, E. N. Mathias, E. P. A. Neto, and A. S. Charao, "Comparaç ao entre plataformas peer-to-peer para desenvolvimento de aplicaç oes de computaç ao distribuıda."
[13] G. Souza, F. V. Mestrando, C. Lima, G. Junior, M. Castro, and A. Sérgio, "Optimal positioning of gprs concentrators for minimizing node hops in smart grids considering routing in mesh networks," in *2013 IEEE PES Conference on Innovative Smart Grid Technologies (ISGT Latin America)*. IEEE, 2013, pp. 1–7.
[14] G. B. d. C. Souza, F. H. T. Vieira, C. R. Lima, G. A. d. J. Deus, M. S. de Castro, S. G. de Araujo, and T. L. Vasques, "Developing smart grids based on gprs and zigbee technologies using queueing modeling–based optimization algorithm," *ETRI Journal*, vol. 38, no. 1, pp. 41–51, 2016.
[15] J. Cai and D. J. Goodman, "General packet radio service in gsm," *IEEE Communications Magazine*, vol. 35, no. 10, pp. 122–131, 1997.
[16] G. Brasche and B. Walke, "Concepts, services, and protocols of the new gsm phase 2+ general packet radio service," *IEEE Communications Magazine*, vol. 35, no. 8, pp. 94–104, 1997.
[17] D. P. Scarpazza, O. Villa, and F. Petrini, "Efficient breadth-first search on the cell/be processor," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 10, pp. 1381–1395, 2008.
[18] V. M. Sanz, "Performance analysis and optimization of parallel best-first search algorithms on multicore and cluster of multicore," *Journal of Computer Science & Technology*, vol. 16, 2016.
[19] M. Bisson, M. Bernaschi, and E. Mastrostefano, "Parallel distributed breadth first search on the kepler architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 2091–2102, 2015.
[20] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 65.
[21] S. H. Permana, K. Y. Bintoro, B. Arifitama, and A. Syahputra, "Comparative analysis of pathfinding algorithms a*, dijkstra, and bfs on maze runner game," *IJISTECH (International J. Inf. Syst. Technol., vol. 1, no. 2, p. 1*, 2018.
[22] J. Monteiro, R. Granada, R. C. Pinto, and R. C. Barros, "Beating bomberman with artificial intelligence," in *Anais do XV Encontro Nacional de Inteligência Artificial e Computacional*. SBC, 2018, pp. 353–364.
[23] R. Libeskind-Hadas and R. Melhem, "Multicast routing and wavelength assignment in multihop optical networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 5, pp. 621–629, 2002.
[24] B. T. Doshi, S. Dravida, P. Harshavardhana, O. Hauser, and Y. Wang, "Optical network design and restoration," *Bell Labs Technical Journal*, vol. 4, no. 1, pp. 58–84, 1999.
[25] Google. Google earth. Acessado em: 24/09/2018. [Online]. Available: https://www.google.com.br/earth/
[26] M. Goldbarg and E. Goldbarg, *Grafos: Conceitos, algoritmos e aplicações*. Elsevier, 2012.
[27] R. Goldschmidt and E. Passos, *Data mining: um guia prático*. Gulf Professional Publishing, 2005.
[28] C. Solomon and T. Breckon, *Fundamentos de processamento digital de imagens: uma abordagem prática com exemplos em Matlab*. Grupo Gen-LTC, 2000.
[29] S. J. Chapman, *MATLAB programming for engineers*. Nelson Education, 2015.
[30] MathWorks. Makers of matlab and simulink. Acessado em: 24/09/2018. [Online]. Available: https://in.mathworks.com/
[31] G. L. Cravo and A. R. S. Amaral, "Um algoritmo grasp com fase de diversificação para problema de layout de facilidades em fila única," in *Annals of the XLIX SBPO-Simpósio Brasileiro de Pesquisa Operacional, Rio de Janeiro, Brazil*, 2017.

Gabriella Lopes Andrade Possui graduação em Ciência da Computação pela Universidade Federal do Pampa (2016) e atualmente é Mestranda em Engenharia Elétrica pela mesma universidade. Possui experiência na área de programação paralela com OpenMP, modelagem e otimização de sistemas matemáticos e desenvolvimento de algoritmos bio-inspirados.

Djeisson Hoffman Thomas Possui graduação em Engenharia Elétrica pela Universidade Federal de Santa Maria (2002), mestrado em Engenharia Elétrica pela Pontifícia Universidade Católica do Rio de Janeiro (2003) e doutorado em Engenharia Elétrica pela mesma instituição (2007). Atualmente é professor efetivo da Universidade Federal do Pampa, Campus Alegrete. Possui experiência na área de Sistemas de Comunicações Ópticas e Instrumentação Optoeletrônica.