UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

IMPLEMENTAÇÃO DE HEURÍSTICAS PARA DETERMINAÇÃO DO CAMINHO DE MENOR CUSTO

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

CHARLES PEREIRA

BLUMENAU, JUNHO/1999

1999/1-10

IMPLEMENTAÇÃO DE HEURÍSTICAS PARA DETERMINAÇÃO DO CAMINHO DE MENOR CUSTO

CHARLES PEREIRA

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

____

Prof. Roberto Heinzle — Orientador na FURB

____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

____

Prof. Roberto Heinzle

____

Prof. Dalton Solano dos Reis

____

Prof. Everaldo Artur Grahl

DEDICATÓRIA

Dedico este trabalho a meus pais que sempre me apoiaram e incentivaram.

AGRADECIMENTOS

Muitas pessoas direta ou indiretamente, ajudaram a tornar possível a conclusão deste trabalho, que marca também, a conclusão do curso de bacharelado em Ciências da Computação. Gostaria de agradecer de um modo especial a:

- meus pais, que sempre me apoiaram e incentivaram durante todo a Faculdade;

- meus colegas, por ajudarem sempre que foi possível e por terem feito parte da minha vida durante estes quatro anos;

- aos professores em geral, principalmente o meu orientador, Roberto Heinzle, pela forma que nos ensinaram e por terem sido amigos e participativos;

- e acima de tudo, a Deus, que me deu talentos para que eu pudesse usá-los.

SUMÁRIO

DEDICATÓRIA iii

AGRADECIMENTOS iv

SUMÁRIO v

LISTA DE FIGURAS viii

RESUMO x

ABSTRACT xi

1 INTRODUÇÃO 1

1.1 MOTIVAÇÃO 1

1.2 OBJETIVOS 2

1.3 ORGANIZAÇÃO DO TEXTO 2

2 TEORIA DOS GRAFOS 3

2.1 CONCEITOS BÁSICOS 3

2.1.1 DEFINIÇÃO DE GRAFO 3

2.1.2 GRAFOS DIRECIONADOS (DIGRAFOS) 4

2.1.3 GRAU DE UM VÉRTICE 5

2.1.4 GRAFOS VALORADOS 5

2.2 PROBLEMA DO MENOR CAMINHO 5

2.2.1 DEFINIÇÃO 5

2.2.2 APLICAÇÕES 6

2.2.2.1 O PROBLEMA DO CAIXEIRO VIAJANTE 6

2.2.2.2 PROBLEMAS DE ENTREGA 6

2.3 REPRESENTAÇÃO DE MAPAS USANDO DIGRAFOS 6

2.4 DEFINIÇÃO COMPUTACIONAL DE GRAFOS 7
2.4.1 MATRIZ DE ADJACÊNCIA 7
2.4.2 MATRIZ DE CUSTO 8
2.4.3 ESTRUTURA DE ADJACÊNCIA 8
2.4.4 A REPRESENTAÇÃO ESCOLHIDA 9
2.5 BUSCA EM GRAFOS 9
2.5.1 BUSCA EM LARGURA 9
2.5.2 BUSCA EM PROFUNDIDADE 10
2.5.3 ALGORITMO DE DIJKSTRA 10
2.5.4 ALGORITMO A* 12
3 HEURÍSTICAS 14
3.1 O JOGO DO OITO 14
3.1.1 BUSCA CEGA 15
3.1.2 BUSCA HEURÍSTICA 18
4 A APLICAÇÃO EXPERIMENTAL 21
4.1 A BASE DE TESTES 21
4.1.1 A CONSTRUÇÃO DA BASE DE TESTES 21
4.2 MEDIDA DE PERFORMANCE DO ALGORITMO 25
4.3 CASOS DE TESTES 26
4.4 PERFORMANCE DA BUSCA CEGA 27
4.5 A PRIMEIRA HEURÍSTICA 28
4.6 A SEGUNDA HEURÍSTICA 31
4.7 VÁRIOS VÉRTICES COMO DESTINO 36
4.7.1 RESOLVENDO O PROBLEMA 37
5 O SISTEMA DESENVOLVIDO 40

5.1 AMBIENTE DE DESENVOLVIMENTO 40
5.2 COMPONENTES DO SISTEMA 40
5.2.1 A BASE DE DADOS 40
5.2.1.1 DICIONÁRIO DE DADOS 42
5.2.2 AS TELAS 44
5.3 OPERAÇÃO DO SISTEMA 45
5.3.1 ENCONTRANDO O MENOR CAMINHO 45
5.3.2 RECURSOS DE NAVEGAÇÃO 46
5.3.3 IDENTIFICANDO O SENTIDO DAS RUAS 47
5.4 PRINCIPAIS PROGRAMAS FONTES 47
5.4.1 BUSCA CEGA 47
5.4.2 A PRIMEIRA HEURÍSTICA 49
5.4.3 HEURÍSTICA DO RETÂNGULO 51
6 CONCLUSÃO 54
6.1 CONSIDERAÇÕES FINAIS 54
6.2 EXTENSÕES 54
REFERÊNCIAS BIBLIOGRÁFICAS 56

LISTA DE FIGURAS

Figura 1 - Exemplo de um grafo.. 4

Figura 2 – Exemplo de um digrafo 4

Figura 3 – Representação de mapas através de digrafos valorados... 7

Figura 4 – Exemplo do Jogo do Oito 14

Figura 5 – Exemplo de um árvore de espaço-estado 15

Figura 6 – Resolução do Jogo do Oito através da busca cega.. 17

Figura 7 – Resolução do Jogo do Oito com o auxílio de heurística 21

Figura 8 – Parte do mapa cedido pela Prefeitura Municipal de Brusque............................... 22

Figura 9 – Tipos de vértices 23

Figura 10 – Parte do mapa de Brusque com indicações dos sentidos das ruas 24

Figura 11 – Mapa de Brusque usado como base de testes 25

Figura 12 - Gráfico da comparação da primeira heurística com a busca cega....................... 30

Figura 13 – Vértices explorados antes de encontrar o vértice destino.................................. 31

Figura 14 – Ilustração da segunda heurística 32

Figura 15 – Gráfico com a comparação da porcentagem de redução do número de vértices explorados de acordo com o valor de h’ 35

Figura 16 – Exemplos de casos de testes bem sucedidos com a Heurística do Retângulo...... 36

Figura 17 – Grafo ilustrando o PMC com vários destinos.. 38

Figura 18– MER da base de dados do sistema... 41

Figura 19 – Diagrama de Contexto... 41

Figura 20 – Diagrama de Fluxo de Dados 42

Figura 21 – Ilustração da interface do protótipo.. 44

Figura 22 – Botões de *zoom* 46

Figura 23 – Botões de deslocamento 46

RESUMO

Este trabalho visa apresentar um estudo sobre Teoria dos Grafos, abordando mais especificamente o seu uso para a representação e resolução de problemas de busca de menor caminho em mapas. Relata ainda, um estudo sobre heurísticas, analisando as suas particularidades e eficiência. Implementa um algoritmo de busca e ainda, um protótipo que se utiliza desse algoritmo e das heurísticas estudadas a fim de resolver o problema do menor caminho.

ABSTRACT

This work intends to present a study about the Graph Theory, more specificly its use for representation and resolution of problems on finding the smallest path in maps. Besides, it brings a study of heuristics analysing their peculiaryties and efficiency. It implements a searching algorith and a prototype that uses this algorith and the heuristics studied to solve the problem on findind the smallest path.

1 INTRODUÇÃO

"Ao contrário de muitos ramos da matemática, nascidos de especulações puramente teóricas, a teoria dos grafos tem sua origem no confronto de problemas práticos relacionados a diversas especialidades e na constatação da existência de estruturas e propriedades comuns, dentre os conceitos relacionados a esses problemas" [BOA79]. "Dentre as técnicas existentes para a solução de problemas algorítmicos em grafos, a busca ocupa lugar de destaque, pelo grande número de problemas que podem ser resolvidos através da sua utilização" [SZW84].

De um modo geral, a área de algoritmos em grafos tem como interesse principal a resolução de problemas, tendo em mente uma preocupação computacional. Ou seja, o objetivo principal é encontrar algoritmos, eficientes se possível, para resolver um dado problema em grafos [SZW84].

Ao lado da teoria dos fluxos em redes, o problema geral de procura de caminhos em grafos é de importância primordial na teoria dos grafos. O Problema do Menor Caminho é o mais importante problema relacionado com a busca de caminhos em grafos, em vista de sua ligação direta com uma realidade encontrada a todo momento [BOA79].

Como exemplo, pode-se citar o problema do caixeiro-viajante, que consiste na determinação da rota de menor custo para uma pessoa que parta de uma cidade e deva visitar diversas outras, passando uma vez e uma só em cada cidade e retornando ao ponto de partida [BOA79]. Uma solução para esse problema seria usar a força bruta, examinando cada permutação do conjunto das cidades e verificar se esta corresponde ou não a um trajeto com as condições exigidas. Mas esta não é uma solução satisfatória do ponto de vista algorítmico, uma vez que só se aplica na prática quando o número de cidades é muito pequeno, de algumas poucas unidades [SZW84].

1.1 MOTIVAÇÃO

Existe uma grande dificuldade em se desenvolver algoritmos para busca em grafos, considerando vários vértices como meta. Dessa forma, para solucionar este problema, não se pode apenas considerar algoritmos matemáticos, mas sim, desenvolver um algoritmo baseado em uma série de heurísticas que dêem "inteligência" ao processamento, tornando-o capaz de chegar a uma solução de forma rápida, mesmo para um grafo com um grande número de vértices.

1.2 OBJETIVOS

O objetivo principal deste trabalho é implementar e avaliar heurísticas que diminuam os cálculos e tornem possível o uso de mapas de grandes cidades, isto é, com milhares de cruzamentos. Depois de implementadas e testadas algumas heurísticas, será tratado o objetivo secundário do trabalho, onde serão mostradas as implicações de resolver o Problema do Menor Caminho com vários vértices como destino.

1.3 ORGANIZAÇÃO DO TEXTO

No capítulo 2, o trabalho começa abordando os conceitos básicos de grafos e como eles podem ser representados e implementados em computador. Em seguida, será explicado o Problema do Menor Caminho e serão analisados os principais algoritmos já formulados para a resolução deste.

O capítulo 3 dedica-se ao estudo de possíveis heurísticas para agilizar a resolução do problema. Será visto o que são heurísticas e como elas podem ser aplicadas no algoritmo.

No capítulo 4, serão feitos testes de performance do algoritmo com e sem as heurísticas encontradas, a fim de tornar possível a avaliação se elas são válidas ou não. Depois, serão abordadas as implicações da resolução do problema tendo mais de um vértice como destino.

O capítulo 5 trata das considerações em relação ao protótipo implementado.

Finalmente, no capítulo 6, encontra-se a conclusão do trabalho.

2 TEORIA DOS GRAFOS

Basicamente, a Teoria dos Grafos trata das relações existentes entre os elementos de um ou mais conjuntos. Tem abrangência grande e muitos problemas que tratam do inter-relacionamento de elementos, seja em química orgânica, eletricidade, transporte, psicossociologia, etc., podem ser estudados com o auxílio de grafos, visto as associações de correspondência entre os elementos do problema, como os átomos de uma molécula, componentes em um circuito elétrico, ruas ligadas por cruzamentos , etc. [BOA79].

2.1 CONCEITOS BÁSICOS

Para uma compreensão do trabalho apresentado, serão mostrados em seguida, os principais conceitos da Teoria dos Grafos. Entendendo a estrutura básica de um grafo, se conseguirá o completo entendimento dos algoritmos que serão apresentados posteriormente.

2.1.1 DEFINIÇÃO DE GRAFO

Um grafo $G(V,E)$ é um conjunto finito não vazio V e um conjunto E de pares não ordenados de elementos distintos de V. Os elementos de V são os vértices e os de E são as arestas de G respectivamente. Cada aresta $e \in E$ será denotada pelo par de vértices $e = (v,w)$ que a forma. Nesse caso, os vértices v e w são os extremos da aresta e, sendo denominados adjacentes. A aresta e é dita incidente a ambos v e w. Duas arestas que possuem um extremo comum são chamadas de adjacentes [SZW84].

Uma aresta incidente a um único vértice, é denominada um laço. Este tipo de aresta, porém, não será considerado neste trabalho, haja visto que não ocorre nos problemas de malha rodoviária, pois não existem ruas que têm seu final no mesmo ponto de seu início. Um exemplo é dado na figura 1.

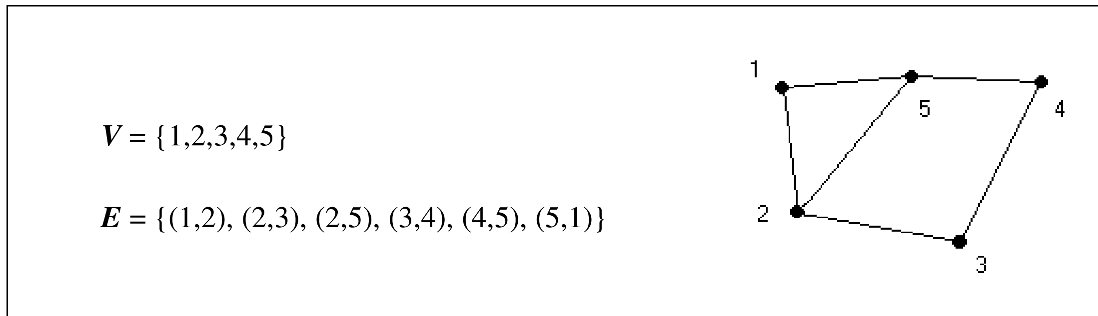


Figura 1 - Exemplo de um grafo

Existe um grande número de tipos de grafos, cada um com suas características e aplicações, entretanto, este trabalho não abordará todos os tipos de grafos, visto que isso é irrelevante para a fundamentação teórica do mesmo. Serão estudadas, a partir de agora, as particularidades de grafos direcionados, que é o tipo que será usado para representar a interligação entre cruzamentos e ruas no sistema de malha rodoviária.

2.1.2 GRAFOS DIRECIONADOS (DIGRAFOS)

Um grafo direcionado, também chamado de digrafo, ***D(V,E)*** é um conjunto finito não vazio ***V*** (vértices) e um conjunto ***E*** (arestas) de pares ordenados de vértices distintos. Portanto, num digrafo, cada aresta ***(v,w)*** possui uma única direção de ***v*** para ***w***. Diz-se também que ***(v,w)*** é divergente de ***v*** e convergente a ***w***. Ao contrário dos grafos não direcionados, os digrafos podem possuir ciclos de comprimento 2, no caso em que ambos ***(v,w)*** e ***(w,v)*** são arestas do digrafo [SZW84]. A figura 2 mostra um exemplo:

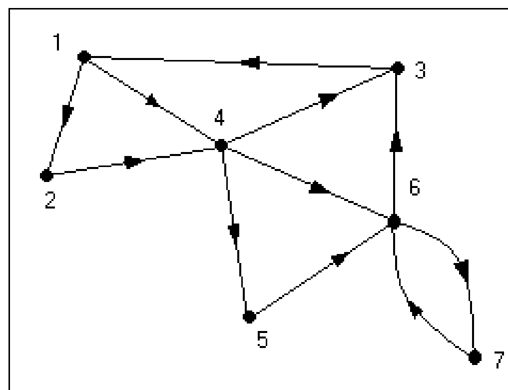


Figura 2 – Exemplo de um digrafo

2.1.3 GRAU DE UM VÉRTICE

O grau de um vértice é o número de arestas que se ligam nele. Nos digrafos, essas arestas podem ser tanto divergentes do vértice ou convergentes a ele. Dessa forma, o grau de um vértice pode ser tanto de entrada como de saída. Seja ***D(V,E)*** um digrafo e um vértice $v \in V$, o *grau de entrada* de v é o número de arestas convergentes a v. O *grau de saída* de v é o número de arestas divergentes de v [SZW84].

2.1.4 GRAFOS VALORADOS

Seja um grafo ***D(V,E)***, para cada aresta $e \in E$, existe um valor associado. O valor pode significar muitas coisas, dependendo do problema sendo representado pelo grafo. No caso de circuitos elétricos, por exemplo, podemos usar o valor 1 para indicar que existe ligação entre dois vértices e 0 quando não existe ligação. No caso de mapas, o valor pode indicar o tempo de deslocamento, a distância ou um custo em unidade monetária.

2.2 PROBLEMA DO MENOR CAMINHO

Será apresentado em seguida o chamado *Problema do Menor Caminho* - PMC.

2.2.1 DEFINIÇÃO

Seja um grafo ***G(V,E)*** e uma função distância ***L*** que associe cada aresta ***(v,w)*** a um número real não negativo ***L(v,w)*** e também um vértice fixo v_0 em ***V***, chamado fonte. O problema consiste em se determinar o caminho de v_0 para um vértice ***vf*** de ***G***, de tal forma que a somatória das distâncias das arestas envolvidas no caminho seja mínima [RAB92].

Como já foi visto, o mapa de uma cidade pode ser representado por um grafo ***G(V,E)***, onde os cruzamentos são os vértices e os segmentos das ruas que ligam estes cruzamentos são as arestas. O PMC consiste em determinar por quais vértices se deverá passar para seguir de um vértice v_0 até um vértice ***vf***, percorrendo-se a menor distância possível.

2.2.2 APLICAÇÕES

O PMC é um importante problema encontrado na Teoria dos Grafos, pela sua aplicação prática. Assim como os grafos podem ser usados para representar matematicamente uma enorme quantidade de problemas, o PMC pode ser usado para resolver estes problemas. A seguir são mostradas algumas aplicações para ele.

2.2.2.1 O PROBLEMA DO CAIXEIRO VIAJANTE

"Suponha que se tenha um mapa de rede rodoviária em que as cidades são representadas por pontos e as estradas por linhas. O mapa, sem dúvida, pode ser considerado um grafo, em que as distâncias são ponderações das arestas, podendo ser consideradas também como custo ou tempo" [RAB92].

O Problema do Caixeiro Viajante, que é chamado de Problema do Carteiro Chinês por alguns autores, consiste em encontrar o caminho de menor custo que o caixeiro viajante deve percorrer, partindo de uma cidade de origem escolhida e visitando todas as outras uma única vez e retornando à cidade de origem.

2.2.2.2 PROBLEMAS DE ENTREGA

O Problema do Caixeiro Viajante é um problema mais teórico que prático, mas existem problemas reais onde se necessita passar por vários vértices e retornar ao vértice de origem com o menor custo possível. Podem ser citados aqui os problemas de entrega de mercadorias, onde se deseja saber qual o caminho que deve ser percorrido a um custo mínimo.

O custo para percorrer certo trecho de um caminho pode ter uma série de variáveis, como o comprimento; as condições da estrada, o nível de tráfego; o número de oficinas mecânicas no trecho, ou qualquer outra variável que possa ser relevante.

2.3 REPRESENTAÇÃO DE MAPAS USANDO DIGRAFOS

Para a representação de mapas rodoviários serão usados digrafos valorados. Cada cruzamento de ruas no mapa é um vértice do grafo, as ruas são representadas pelas arestas. Se a rua tiver mão única, diz-se que ela possui ciclo de comprimento 1, ou seja, a orientação da

aresta existe em apenas um sentido. Já as ruas com mão dupla, possuem ciclos de comprimento 2.

Partindo-se do princípio que não existem ruas sem saída de mão única, todos os vértices, que representam os cruzamentos, tem tanto o *grau de entrada* quanto o *grau de saída* de pelo menos uma unidade. Em outras palavras, na representação de mapas com grafos, todos os vértices têm pelo menos uma aresta de entrada e uma de saída.

A figura 3 ilustra um exemplo de um mapa representado através de um digrafo valorado. Note que "caminhando" pelo grafo, é possível chegar a qualquer vértice, não importando qual o vértice de origem. As arestas ***(6,7)*** e ***(7,6)*** representam uma rua sem saída, na verdade, a sua saída é a própria entrada. As ruas sem saída não se caracterizam como um problema, pois a rua sempre terá mão dupla. Esse é o motivo da sua representação através de duas arestas.

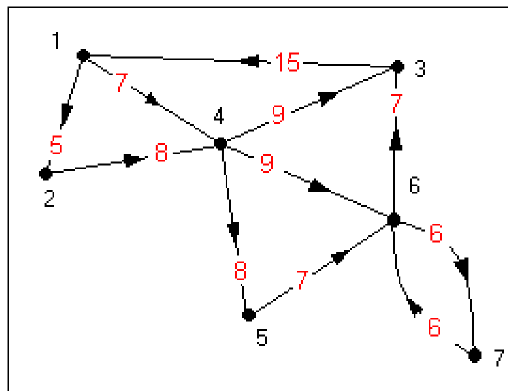


Figura 3 – Representação de mapas através de digrafos valorados

2.4 DEFINIÇÃO COMPUTACIONAL DE GRAFOS

A representação mais familiar de um grafo é através do desenho de pontos e linhas. Já em um computador, o grafo pode ser representado de diversas maneiras. A eficiência do algoritmo vai depender da escolha certa de como representar um grafo [RAB92].

2.4.1 MATRIZ DE ADJACÊNCIA

Dado um grafo $G(V,E)$, a matriz de adjacência $A = [a_{ij}]$ é uma matriz n x n tal que

a_{ij} = 1 se e somente se existe $(v_i,v_j) \in E$

ou

a_{ij} = 0 caso contrário.

A matriz de adjacência só se aplica a grafos não direcionados [RAB92]. Em virtude disso, essa representação não será usada para representar o problema proposto, só foi comentada para ilustrar uma maneira possível de representação de grafos.

2.4.2 MATRIZ DE CUSTO

Um grafo, no qual um número w_{ij} está associado a cada aresta, é denominado de grafo valorado e o número w_{ij} é chamado o custo da aresta. Em redes de comunicação ou transporte estes custos representam alguma quantidade física, tal como distância, eficiência, capacidade da aresta correspondente, etc. [RAB92].

Alguns algoritmos usam o processamento através de matrizes de custo. Um exemplo é o algoritmo de Floyd, que usa três matrizes para determinar qual é o menor caminho entre um vértice origem e um destino. A primeira é a matriz de custos **D[n,n]**, onde os laços – vértice destino é o mesmo que o vértice origem – possuem custo zero e a não existência de aresta, é atribuído custo infinito. Considerando a matriz de custos **D[n,n]**, o algoritmo produzirá uma matriz **A[n,n]** com as menores distâncias entre os vértices, e ainda uma matriz **R[n,n]** que fornece a rota a ser seguida para ir do vértice origem até o vértice destino. Para construir a matriz **A[n,n]**, o algoritmo constrói sucessivamente n matrizes a partir de **D[n,n]**, através de modificações efetuadas de acordo com uma expressão matemática específica [RAB92].

2.4.3 ESTRUTURA DE ADJACÊNCIA

Um vértice y em um grafo é chamado um sucessor de um vértice x, se existe uma aresta dirigida de x para y; o vértice x é denominado um antecessor de y. Um grafo pode ser descrito por sua estrutura de adjacência (adj), isto é, pela lista de todos os sucessores de cada vértice. Portanto, para cada vértice v, adj(v) consiste de uma lista de todos os sucessores do vértice v. Em grafos valorados, além da lista de todos os sucessores do vértice v, a estrutura de adjacência contém também o valor atribuído à aresta. Nesse tipo de representação, não é usado matrizes, mas sim listas.

2.4.4 A REPRESENTAÇÃO ESCOLHIDA

Como o objetivo do trabalho é usar grafos que representem cidades com milhares de cruzamentos, se torna inviável o uso da matriz de custo, pois a quantidade de memória alocada seria bastante grande. Assim sendo, a forma de armazenamento utilizada neste trabalho é a estrutura de adjacência.

2.5 BUSCA EM GRAFOS

Dentre as técnicas existentes para a solução de problemas algorítmicos em grafos, a busca ocupa lugar de destaque, pelo grande número de problemas que podem ser resolvidos através da sua utilização [SZW84]. A busca consiste em localizar um vértice dentro de um grafo, traçando o caminho percorrido para se deslocar de um vértice inicial até o vértice que se deseja procurar. Os dois métodos mais comuns que existem são a Busca em Largura e a Busca em Profundidade.

2.5.1 BUSCA EM LARGURA

O algoritmo de Busca em Largura é uma proposta que trabalha sob o critério FIFO (*First In First Out*, isto é, o primeiro que entra é o primeiro que sai). É também denominada de busca em amplitude, busca em nível e "breadth-first" [RAB95].

Funciona da seguinte forma: "Aplica-se todos os operadores possíveis à raiz (isto é, ao estado inicial), em seguida, a todos os filhos da raiz (da esquerda para a direita), depois, a todos os "netos" da raiz (da esquerda para a direita) e assim por diante. A busca cessa ao se descobrir o nó objetivo." [COL88].

Existem duas filas de trabalho usadas pela busca em largura. A fila de abertos, que guarda todos os vértices do grafo a serem explorados; e a fila de fechados, onde se encontram todos os vértices que já foram explorados. O algoritmo mostrado no Quadro 1 ilustra de forma sucinta este tipo de busca.

```
      Cria duas filas (abertos e fechados);
      Inicializa a fila de abertos (nó início);
      Enquanto abertos não estiver vazia, faz:
    Remove um nó da fila abertos e chama de X
         Se X é a meta, termina com sucesso
         Senão busca todos os filhos de X
           Se algum dos filhos de X é a meta,          termina com sucesso
           Senão dispensa os filhos de X que já estão em abertos ou fechados
           Coloca na fila de abertos os filhos remanescentes de X
         Coloca X na fila de fechados
      Fim enquanto
```

Quadro 1 – Algoritmo de busca em largura

Neste algoritmo, todos os vértices de certo nível da árvore são examinados antes do nível abaixo. Se existe uma solução e se o grafo é finito, então por este método ela certamente será encontrada.

2.5.2 BUSCA EM PROFUNDIDADE

O algoritmo de Busca em Profundidade é uma proposta que trabalha sob o critério LIFO (*Last In First Out*, isto é, o último que entra é o primeiro que sai). É também denominada de primeiro-em-profundidade ou “depth-first” [RAB95].

Quanto à estrutura do algoritmo, a única diferença em relação à busca em largura, é a utilização de uma pilha ao invés de fila. Esta técnica explora o caminho para o objetivo, dando preferência aos nós que estão mais distantes da raiz da árvore de busca. Funciona bem, quando as soluções são total e igualmente desejadas ou quando anteriormente foi feita uma varredura, detectando direções incorretas [FUR73].

2.5.3 ALGORITMO DE DIJKSTRA

O algoritmo de Dijkstra, publicado e 1950, se propõem a resolver o PMC e tem a seguinte idéia, conforme demonstrado por [RAB92]:

- Considere o digrafo G(V, E), uma fonte v_0 e uma função L que associe a cada aresta a um número real não negativo, isto é,

$$\mathbf{L}(\boldsymbol{v}_\mathbf{i},\boldsymbol{v}_\mathbf{j}) = \begin{cases} \infty & \text{, se não existe a aresta } (\boldsymbol{v}_\mathbf{i},\boldsymbol{v}_\mathbf{j}) \\ 0 & \text{, se } \boldsymbol{v}_\mathbf{i} = \boldsymbol{v}_\mathbf{j} \\ \text{custo} & \text{, se } \boldsymbol{v}_\mathbf{i} \neq \boldsymbol{v}_\mathbf{j} \text{ e existe a aresta } (\boldsymbol{v}_\mathbf{i},\boldsymbol{v}_\mathbf{j}) \end{cases}$$

- Constrói-se um conjunto S, que contém os vértices v_i's cujo comprimento mínimo de v_0 a cada v_i, seja conhecido;

- A cada passo se adiciona ao conjunto S o vértice w pertencente a V-S tal que o comprimento do caminho v_0 a w, seja menor que o correspondente de qualquer outro vértice de V-S;

- Pode-se garantir que o caminho mínimo de v_0 a qualquer vértice v em S contém somente vértices pertencentes a S.

O desenvolvimento do algoritmo de Dijkstra é feito da seguinte forma:

```
     INÍCIO
          S ← {v0};
          D[v0] ← 0;
          Para cada v ∈ V – {v0} faça D[v] ← L(v0, v);
          Enquanto S ≠ V faça
               Escolha o vértice w ∈ V – S tal que D[w]
seja mínimo;
               Coloque w em S, isto é, faça S ← S ∪{w};
               Para cada v ∈ V – S faça
                    D[v] ← MIN (D[v], D[w] + L(v, w));
     FIM
```

Quadro 2 – Algoritmo de Dijkstra

2.5.4 ALGORITMO A*

O algoritmo A* é uma derivação do algoritmo de Dijkstra, na verdade, eles são iguais em sua essência. A diferença é que o A* é acrescido de um critério de busca, definido pela função heurística f' = g + h', que calcula o mérito de cada vértice gerado [RAB92].

A função g é a estimativa do custo para ir do vértice inicial até o atual; a função h é a estimativa do custo adicional para ir do vértice atual até o meta. Portanto, a função f' = g + h' representa uma estimativa de custo para ir do vértice inicial até o vértice meta, ao longo do caminho que gerou o vértice atual. Se houver mais que um caminho que gerou o vértice, então o algoritmo registrará o melhor.

A idéia do algoritmo é expandir um vértice a cada passo, até gerar um vértice que corresponda ao vértice meta. Em cada passo, escolhe-se o vértice mais promissor daqueles gerados mas não expandidos. Gera os sucessores do vértice escolhido, e aplica a função heurística para em seguida adicioná-los à lista de vértices abertos, após conferir se algum deles já foi gerado anteriormente. Ao fazer esta conferência, fica garantido que cada vértice apareça apenas uma vez, embora muitos vértices possam apontar para ele como um sucessor.

Para o desenvolvimento do algoritmo A* é necessária a utilização de duas listas de vértices:

- ABERTO – vértices que foram gerados e tiveram a função heurística a eles aplicada, e ainda não foram examinados, isto é, vértices que tiveram seus sucessores gerados;

- FECHADO – vértices que já foram examinados.

A algoritmo A* segue os seguintes passos, conforme o quadro 3.

```
Inicie com ABERTO contendo apenas o vértice inicial
     Fixe o valor g desse vértice para zero e o valor
     de h' para o arbitrado.
     Faça a lista FECHADO ← ∅;
     Faça P[v] ← 1 [apontador P aponta para o seu
     melhor antecessor].
     Enquanto o vértice meta não for encontrado, faça
     Se não houver vértice em ABERTO então pare com
     fracasso.
       Senão
          Considere o vértice v ∈ ABERTO com menor
          valor f';
          Retire v da lista ABERTO;
          Coloque v em FECHADO;
          Se v for meta então pare, a solução foi
          encontrada.
          Senão
           Gere os sucessores de v;
           Para cada sucessor w de v faça
             g(w) ← g(v) + d(v,w);
             Se w está em ABERTO então
               Coloque w na lista de sucessores de v;
               Calcule f'(w);
               Se o novo valor de f'(w) for melhor do
               que seu valor anterior então
                 Faça P(w) ← v;
                 Coloque W na lista FECHADO;
                 Guarde o novo valor de f'(w);
```

Quadro 3 - Algoritmo A*

3 HEURÍSTICAS

Heurísticas são técnicas simples, apoiadas em conhecimentos empíricos ou não, que são capazes de direcionar os cálculos para a(s) meta(s). Um exemplo de heurística é dado por [COL88]:

"..se você deseja ir da sua casa para o local de trabalho em um tempo mínimo, pode concluir, baseado em experiências anteriores, que é mais rápido ir de carro, de segunda-feira a quinta-feira, mas na sexta-feira é melhor utilizar o metrô. Esses pressentimentos ou intuições que usamos no dia-a-dia nos auxiliam bastante, mesmo que falhem algumas vezes."

Para a resolução de problemas computacionais complexos, ou seja, que demandam um grande tempo de cálculo, é extremamente importante que se ache heurísticas aplicáveis a fim de reduzir esse tempo de cálculo.

3.1 O JOGO DO OITO

Será utilizado como exemplo o Jogo do Oito e será mostrado como uma heurística simples pode reduzir drasticamente os cálculos. O Jogo do Oito consiste de uma matriz 3x3 com números de um a oito alocados de forma aleatória e uma célula da matriz vazia. O objetivo do jogo é deixar os números posicionados de alguma maneira pré determinada, como indicado na figura 4.

1	2	3
8		4
7	6	5

Figura 4 – Exemplo do Jogo do Oito

A forma que será usada para resolver o problema é a representação espaço-estado, onde um estado é um valor de algum tipo determinado de dados. Começa-se com um estado inicial e tenta-se obter um estado objetivo. Um operador é usado para transformar um estado em outro. O espaço-estado é a coleção de todos os estados que podem ser obtidos a partir do

estado inicial através da aplicação de alguma combinação de operadores. Uma solução para um problema é um caminho de operadores que transforma um estado inicial e um estado objetivo [COL88].

Existem nove valores possíveis para qualquer célula: um número de 1 a 8 ou vazio. Como existem 9 células no jogo, o número total de estados possíveis é 9 fatorial, ou 362880 estados. A figura 5 mostra um exemplo de uma árvore de espaço-estado até a profundidade 3.

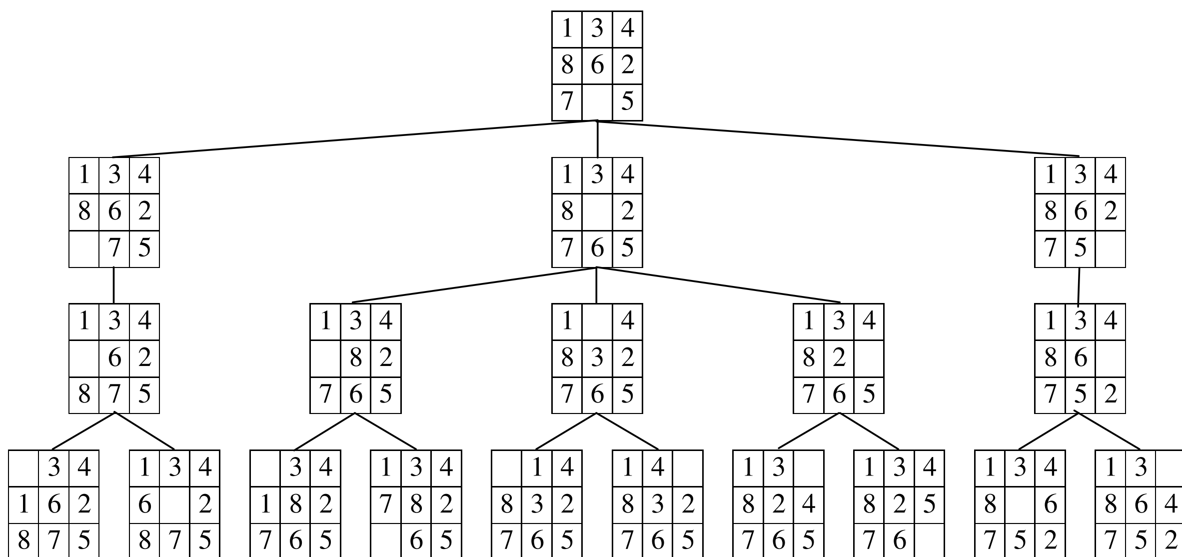


Figura 5 – Exemplo de um árvore de espaço-estado. Fonte: [COL88]

3.1.1 BUSCA CEGA

Quando não se tem uma heurística para determinar como o problema será resolvido, deve-se usar a busca cega, isto é, a ordem em que os operadores são aplicados para gerar outros estados, é sempre a mesma.

Será mostrado a seguir como usar a busca cega para resolver o Jogo do Oito. Para isso, é preciso determinar quais serão os operadores capazes de alterar os estados e em qual ordem eles serão aplicados. O critério de busca que será aplicado é a busca em largura, e os operadores serão aplicados sobre os estados gerados sempre da esquerda para a direita.

Os operadores possíveis são mover a célula vazia para cima **(C)**, baixo **(B)**, esquerda **(E)** ou direita **(D)**. Eles sempre serão aplicados na ordem **(C)**, **(B)**, **(E)** e **(D)**.

Serão considerados os seguintes estados:

- Estado inicial:

1	2	3
7	8	4
6	5	

- Estado final:

1	2	3
8		4
7	6	5

Cada vez que for criado um novo estado aplicando-se um operador, deve-se verificar se o estado já foi criado anteriormente. Portanto, em toda a árvore espaço-estado não podem haver estados repetidos.

Quando um operador não puder ser aplicado, tanto por sua impossibilidade, como por não poder gerar um estado repetido, o novo estado que seria criado não será mostrado.

A figura 6 mostra a resolução do problema. Repare que foram gerados 30 estados para se chegar à solução.

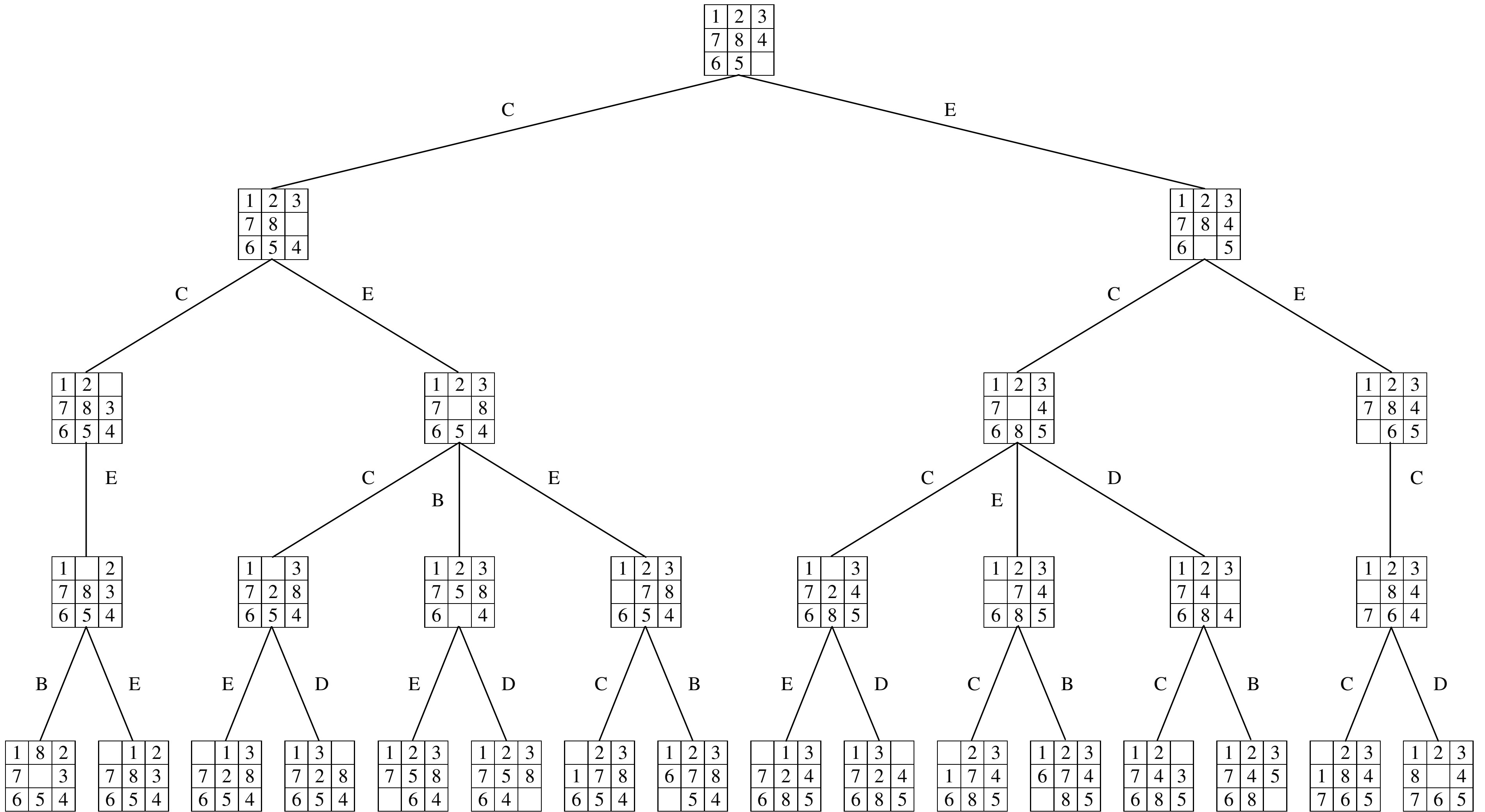


Figura 6 – Resolução do Jogo do Oito através da busca cega

3.1.2 BUSCA HEURÍSTICA

Nesse pequeno exemplo visto na Figura 6, o estado inicial estava muito próximo do estado final desejado, pois somente quatro movimentos eram necessários para encerrar o jogo, mas mesmo assim, foram gerados 30 estados. Agora serão usados os mesmos estados inicial e final para mostrar como uma heurística pode reduzir o número de passos efetuados pelo algoritmo.

É criada uma função heurística, cujo resultado determina o custo da ação, que será usado para determinar qual será o próximo estado a ser explorado. Os operadores continuarão a ser aplicados na mesma ordem, a única diferença, é que o estado escolhido para ser explorado não vai mais seguir necessariamente a ordem da esquerda para direita, mas sim será aquele que tiver o menor valor da função heurística. Quando houver vários estados a serem explorados com o mesmo valor da função heurística, os critérios de seleção, ou desempate, serão respectivamente o que tiver maior profundidade e o que estiver mais à esquerda.

A função heurística é dada por: ***f'(n) = g(n) + h'(n)***, onde:

a) ***n*** = identificador do estado;

b) ***g(n)*** = o custo para ir do estado inicial até o estado ***n***;

c) ***h'(n)*** = valor da heurística para ir do estado ***n*** até o estado meta.

Nesse caso, o valor dado a ***g(n)*** será dado pela profundidade de ***n***.

A valor de ***h'(n)*** pode ser o somatório do valor de várias heurísticas, quando elas existem.

Como o objetivo do jogo é mudar as posições dos números nas células a fim de posicioná-los de acordo com o estado final, quanto mais números já estiverem na posição correta, significa que esse estado está mais próximo da solução. Portanto, para cada número fora da posição final, o valor ***h'(n)*** será acrescido de uma unidade. Assim, o estado que tem mais números fora da posição final, terá um maior valor da função heurística, e a prioridade será dada para aqueles estados mais promissores.

Exemplo:

- Estado atual:

1	2	3
7	4	8
6	5	

- Estado final:

1	2	3
8		4
7	6	5

No estado atual, o valor de ***h'(n)*** = 6, pois tanto a célula vazia, como os números 7, 4, 8, 6 e 5 estão fora da posição final. A figura 7 mostra a resolução do mesmo problema da Figura 6, só que usando a função heurística mostrada agora. Repare que foram gerados 10 estados apenas para chegar à solução, contra trinta através da busca cega.

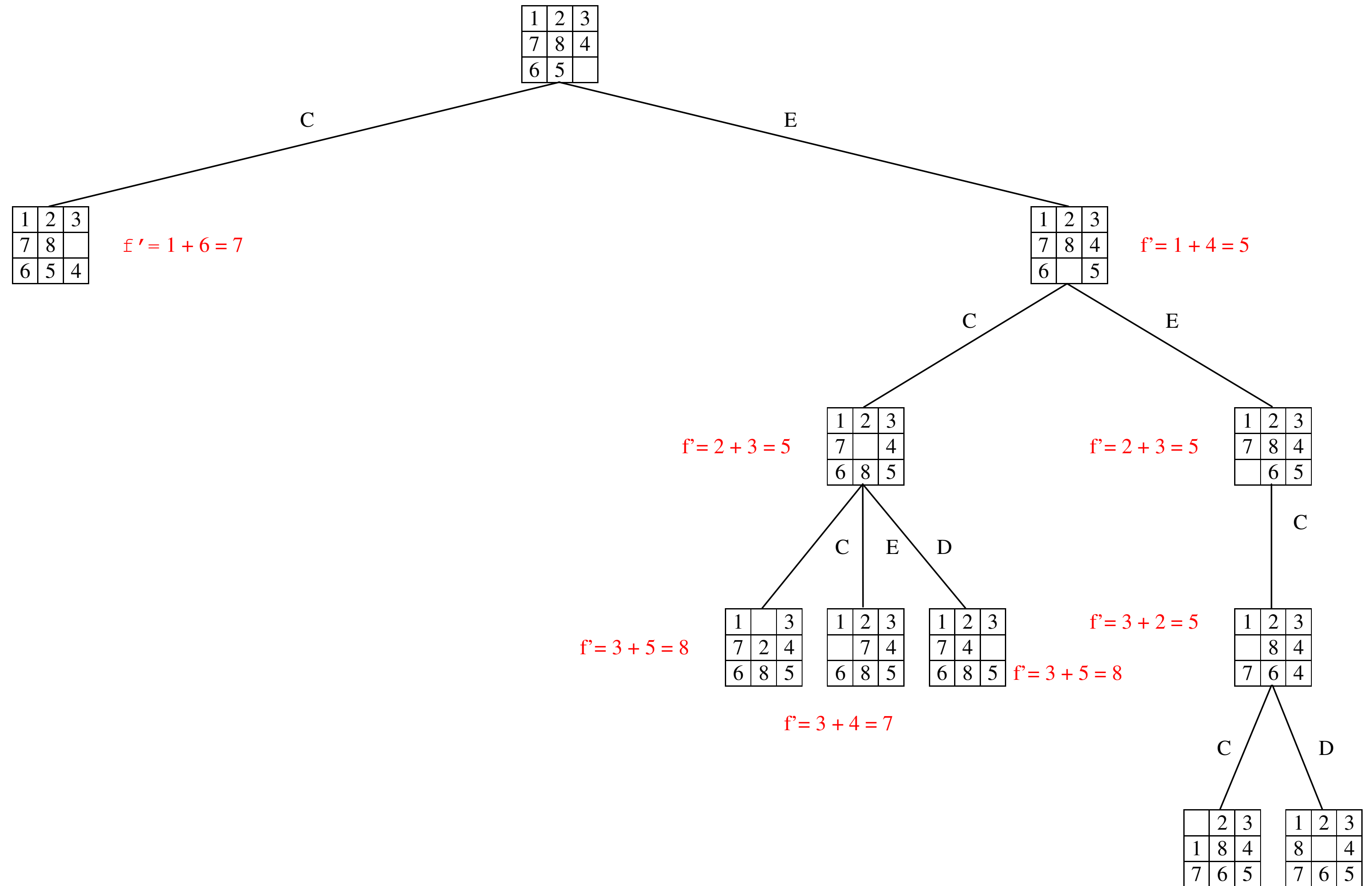


Figura 7 – Resolução do Jogo do Oito com o auxílio de heurística

4 A APLICAÇÃO EXPERIMENTAL

4.1 A BASE DE TESTES

Para cumprir o primeiro objetivo do trabalho, que é a implementação e avaliação de heurísticas, se faz necessário um protótipo que implemente um algoritmo de busca e uma base de testes. Mais adiante está reservado um capítulo dedicado somente ao protótipo, explicando como ele funciona, ambiente de desenvolvimento, etc. Quanto a base de testes, não seria interessante utilizar uma pequena, ou seja, com poucos cruzamentos, já que o objetivo é permitir que se trabalhe com cidades que possuam milhares de ruas. Dessa maneira, utilizou-se como base de testes parte do mapa da cidade de Brusque, no estado de Santa Catarina.

4.1.1 A CONSTRUÇÃO DA BASE DE TESTES

Seria uma tarefa árdua fazer a conversão do mapa de uma cidade para um grafo, cadastrando-se os vértices e arestas manualmente. Dessa forma, procurou-se auxílio com a Prefeitura Municipal de Brusque, que cedeu o desenho do mapa feito no software AutoCAD R13. No mapa cedido pela prefeitura, não existe uma diferenciação para as ruas com mão única e mão dupla, existe apenas o desenho delas. A figura 8, mostra parte do mapa.

O motivo da escolha da cidade de Brusque como base de testes se deve ao fato da disponibilidade dos dados, que em geral, são difícil de encontrar. Além disso, o mapa da cidade se encaixe nos requisitos especificados no objetivo do trabalho, ou seja, possui milhares de cruzamentos, o que torna esta base de dados ideal para a realização de testes.

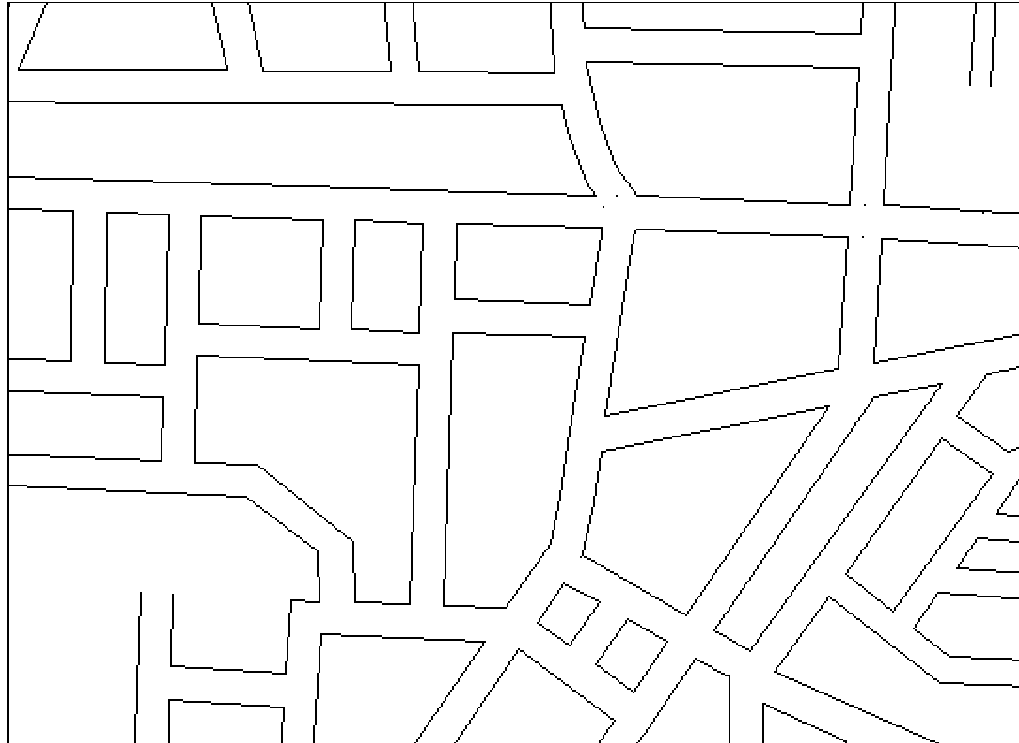


Figura 8 – Parte do mapa cedido pela Prefeitura Municipal de Brusque

Para transformar um mapa em um grafo, deve-se inicialmente determinar quais são os vértices desse grafo. Algumas considerações quanto à determinação dos vértices devem ser feitas, para isso, a figura 9 será usada como referência.

Um vértice existe em três situações:

- cada vez que uma rua se cruzar com outra, como pode ser visto nos vértices 1, 4 e 6;
- no extremo de uma rua, como no vértice 5;
- e quando se necessita mostrar as curvas feitas pelas ruas, como nos vértices 2 e 3. Esses vértices poderiam ser ignorados, criando-se uma aresta apenas, com a distância necessária para ir do vértice 1 ao 4, mas como o protótipo construído precisa mostrar as informações na tela em forma de desenho para o usuário poder entendê-las, foi necessário criar vértices deste tipo.

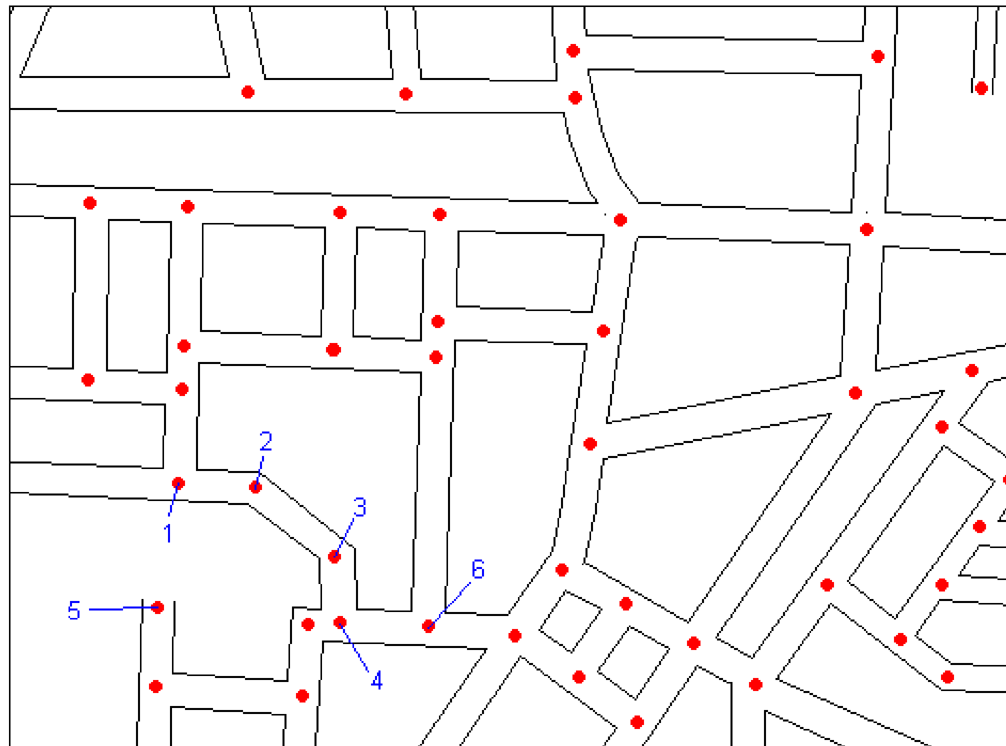


Figura 9 – Tipos de vértices

Depois de todos os vértices terem sido criados, é a vez das arestas. Como é extremamente relevante ao trabalho considerar o sentido das ruas, devem ser criadas sempre duas arestas quando a rua tiver mão dupla. Foram criadas linhas ligando os vértices previamente definidos, a fim de representar o sentido de cada trecho de rua. Entende-se por trecho de rua, o pedaço da rua que liga um vértice a outro. Vale lembrar que os sentidos das ruas usados no trabalho não condizem com a realidade.

A figura 10 mostra a mesma área do mapa mostrada na figura 9, só que agora com as indicações dos sentidos das ruas. Note que quando um trecho de rua tem sentido único, as setas apontam sempre para o mesmo lado, se tiver mão dupla, então existe uma seta apontando para cada lado.

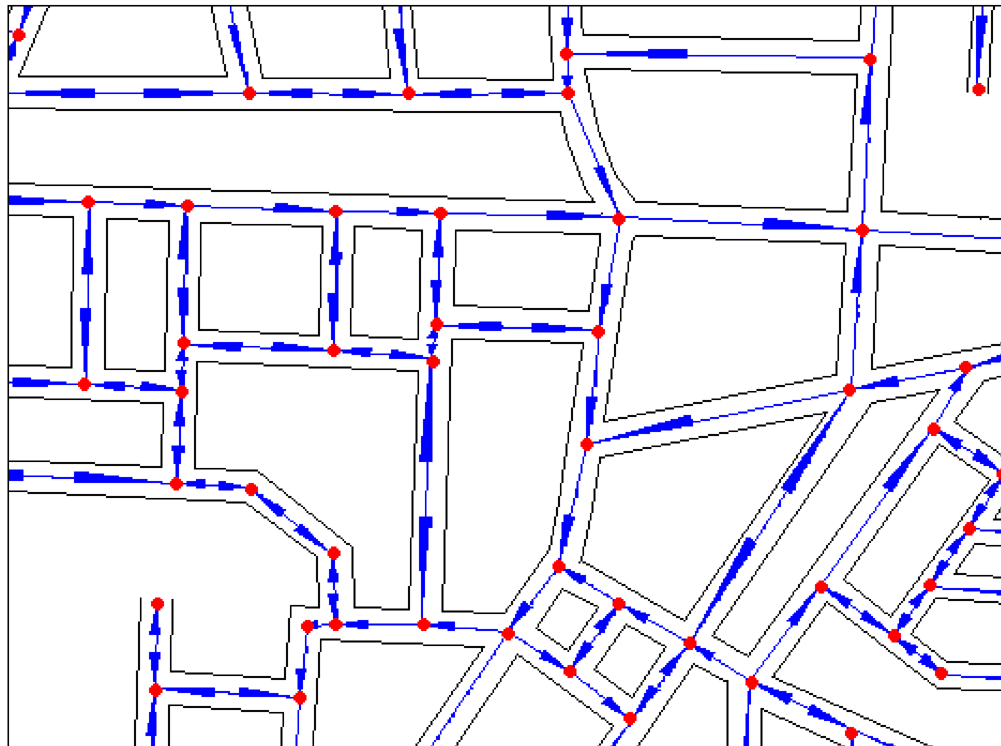


Figura 10 – Parte do mapa de Brusque com indicações dos sentidos das ruas

Cada seta desenhada no mapa eqüivale à uma aresta no grafo. Mas como é preciso usar digrafos valorados, só os dados vistos até agora não são suficientes para deixar a base de testes completa, pois ainda faltam os valores relativos a cada aresta. Adotou-se a distância entre dois vértices como sendo o valor de cada aresta. Não foi necessário cadastrar o comprimento de cada aresta do grafo, pois foi possível calcular a distância entre os vértices quando foi gerada a lista de adjacências.

Como foi mostrado no capítulo 2, os vértices de um grafo são numerados (ver figura 1), assim cada cruzamento do mapa recebeu um código e, dessa forma, todos os dados necessários para se construir a lista de adjacências ficaram completos. A digitalização dos dados resultou em um grafo com 1313 vértices e 2940 arestas. As informações relativas ao comprimento dos trechos das ruas foi fornecida em metros. Para ter uma noção do tamanho da base de teste construída, a figura 11 mostra toda a área digitalizada.

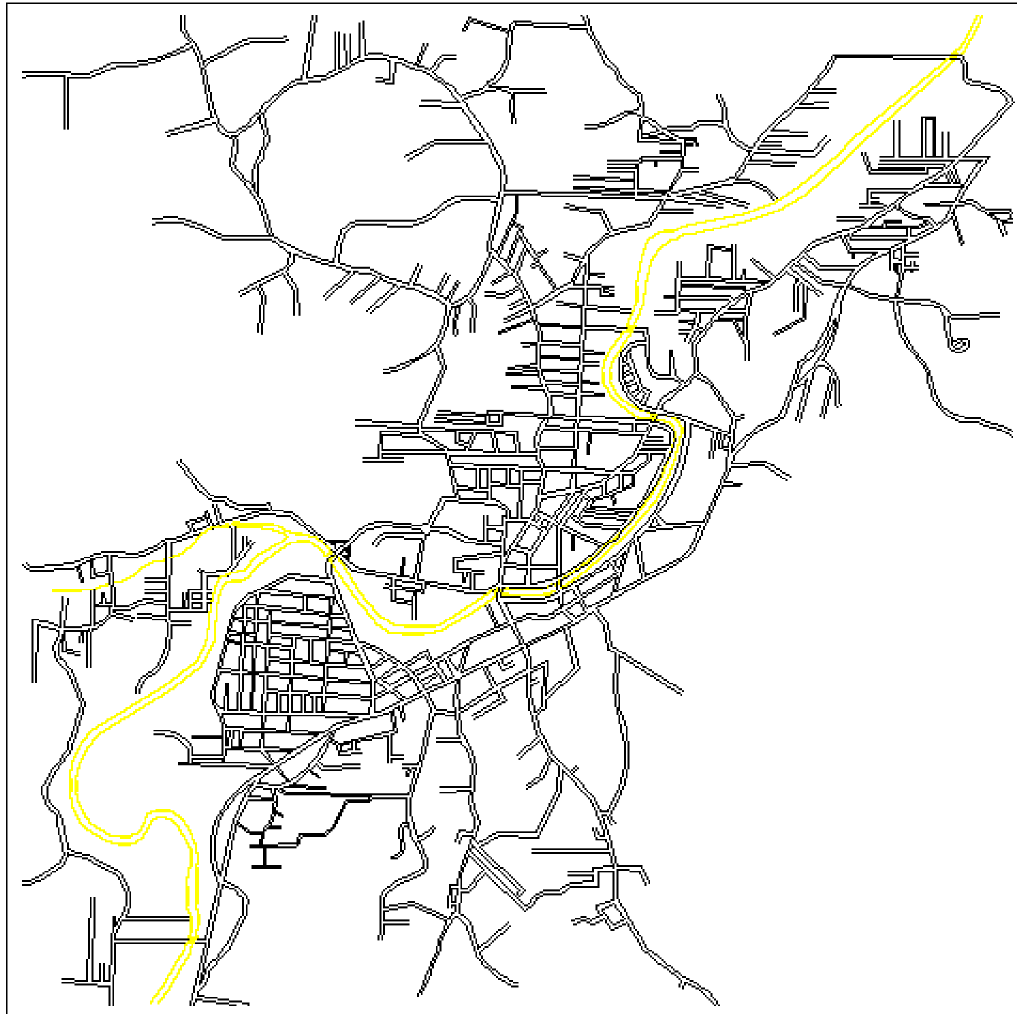


Figura 11 – Mapa de Brusque usado como base de testes

4.2 MEDIDA DE PERFORMANCE DO ALGORITMO

Com a base de testes montada, pode-se iniciar a implementação e avaliação de possíveis heurísticas para melhorar o processamento do algoritmo. Independente de quais heurísticas que serão criadas, para serem avaliadas, é necessário saber como é a performance do protótipo construído sem considerá-las, isto é, usando a busca cega. Portanto, é necessário adotar um padrão de avaliação de performance.

O tempo necessário para resolver o PMC, está ligado diretamente ao número de vértices envolvidos no cálculo. Dessa forma, quanto maior o número de vértices envolvidos,

maior será o tempo consumido para resolver o PMC. A função das heurísticas será dar prioridade para os vértices que têm maior chance de conduzirem à meta.

O algoritmo implementado no protótipo é o A*, que foi mostrado anteriormente. A princípio, as heurísticas serão formuladas para trabalharem com o algoritmo na sua essência, ou seja, tendo apenas um vértice como meta. As heurísticas formuladas nesta etapa, podem não ser aplicáveis ou eficientes quando o algoritmo for modificado para trabalhar com vários vértices de destino, mas esta dúvida será esclarecida posteriormente, quando será tratado o segundo objetivo do trabalho.

Como foi visto, o A* trabalha com uma lista de vértices que foram explorados durante o cálculo, que é a chamada lista *fechado* e uma lista de vértices a explorar, a lista *aberto*. Cada vez que um vértice é retirado da lista *aberto* para ser explorado, o algoritmo analisa todos os vértice adjacentes a ele, para saber se eles serão adicionados na lista de *aberto* ou não. O número de vértices adjacentes ao vértice sendo explorado é variável, isto é, o seu grau varia de acordo com o tamanho do cruzamento. Como o grau é variável, o tempo consumido para avaliar cada vértice também é. Assim sendo, não pode-se considerar o número de vértices na lista fechados como uma medida de eficiência exata. Porém, não é possível fazer o algoritmo deixar de explorar todos os vértices adjacentes ao vértice que está sendo explorado, isto é inerente a ele. Assim, será usado o número de nós explorados durante o processamento do algoritmo como forma de avaliação das heurísticas. Quanto menor for este número, melhor será a heurística.

4.3 CASOS DE TESTES

Os casos de testes serão responsáveis por mensurar a performance do algoritmo. Foram formulados sessenta casos de testes, onde procurou-se definir problemas que, na sua maioria, resultassem em um caminho atravessando grande parte da cidade. Isso se deve ao fato que quanto mais distantes estão os vértices de origem e destino, maior será o número de vértices considerados nos cálculos.

A idéia é aplicar os casos de testes usando primeiramente a busca cega. Assim, serão tabelados os números de vértices envolvidos sem a aplicação de heurísticas. Para cada heurística que for implementada, serão aplicados os casos de testes e os resultados destes

serão comparados com àqueles obtidos usando-se a busca cega. Com isso, será possível mensurar a qualidade de cada heurística apresentada.

4.4 PERFORMANCE DA BUSCA CEGA

Na seção 3.1.2, foi mostrada a comparação de performance da busca cega com uma heurística para o Jogo do Oito. A performance da busca cega para os casos de testes apresentados servirá como comparação para as heurísticas apresentadas posteriormente. O código fonte do algoritmo utilizado encontra-se na seção 5.4.1. A tabela 1 mostra o número de vértices explorados em cada caso de teste usando a busca cega.

N° TESTE	EXPLORADOS	N° TESTE	EXPLORADOS
1	1258	31	968
2	1309	32	1039
3	1283	33	1228
4	1091	34	1310
5	524	35	1230
6	1306	36	1275
7	1019	37	514
8	1150	38	396
9	1180	39	448
10	1307	40	1244
11	364	41	1298
12	534	42	1250
13	1172	43	636
14	747	44	736
15	556	45	558
16	1031	46	387
17	1156	47	949
18	1311	48	835
19	1271	49	1100
20	1153	50	944
21	983	51	159
22	876	52	176
23	563	53	228
24	199	54	152
25	1301	55	102
26	1229	56	94
27	1246	57	92
28	1139	58	67
29	609	59	71
30	897	60	815

Tabela 1 – Vértices explorados com a busca cega

4.5 A PRIMEIRA HEURÍSTICA

Para tabelar a performance da busca cega, foi usada a implementação do algoritmo A* sem heurística. Este algoritmo tem uma heurística que baseia-se na provável distância do vértice que está sendo explorado no momento até o vértice meta. Como o algoritmo em si explora sempre os vértices de menor custo, à medida que são efetuados os passos do algoritmo, ele vai explorando os vértices ao redor do vértice origem. Para ele não importa se os vértices explorados estão entre os vértices origem e destino ou se estão do lado aposto. Por esse motivo, muitos vértices são explorados sem terem a mínima chance de fazer parte do menor caminho.

Se a heurística inerente ao algoritmo for aplicada, então uma quantidade bem menor de vértices será explorada, pois o custo será a soma da distância para se deslocar do vértice origem até o vértice atual, acrescido da provável distância para se deslocar do vértice atual até o destino. Assim, os vértices que se encontram no lado oposto do destino, terão um custo alto, pois estão mais distantes deste. Como valor da heurística, foi dada a distância geométrica entre o vértice sendo explorado e o vértice destino. A distância foi calculada através das coordenadas euclidianas de cada vértice, usando-se como valor distância obtida traçando-se uma linha reta entre os dois vértices.

A tabela 2 mostra respectivamente o número de vértices explorados usando a busca cega, o número de vértices explorados usando a primeira heurística e a porcentagem de redução do número de vértices explorados. A porcentagem média de redução do número de vértices explorados foi de 61%. Logo abaixo, a figura 12 traz um gráfico que mostra a mesma comparação, sem mostrar as porcentagens.

N°	Explor. Busca Cega	Explor. 1ª Heurística	% Redução
1	1258	557	56%
2	1309	684	48%
3	1283	408	68%
4	1091	433	60%
5	524	253	52%
6	1306	953	27%
7	1019	347	66%
8	1150	465	60%
9	1180	730	38%
10	1307	954	27%
11	364	92	75%
12	534	169	68%
13	1172	375	68%
14	747	471	37%
15	556	141	75%
16	1031	147	86%
17	1156	396	66%
18	1311	927	29%
19	1271	799	37%
20	1153	468	59%
21	983	289	71%
22	876	390	55%
23	563	297	47%
24	199	53	73%
25	1301	607	53%
26	1229	417	66%
27	1246	271	78%
28	1139	232	80%
29	609	256	58%
30	897	302	66%

N°	Explor. Busca Cega	Explor. 1ª Heurística	% Redução
31	968	392	60%
32	1039	292	72%
33	1228	395	68%
34	1310	1057	19%
35	1230	293	76%
36	1275	634	50%
37	514	201	61%
38	396	90	77%
39	448	88	80%
40	1244	487	61%
41	1298	594	54%
42	1250	272	78%
43	636	180	72%
44	736	145	80%
45	558	134	76%
46	387	69	82%
47	949	191	80%
48	835	132	84%
49	1100	204	81%
50	944	374	60%
51	159	36	77%
52	176	30	83%
53	228	53	77%
54	152	69	55%
55	102	24	76%
56	94	12	87%
57	92	21	77%
58	67	22	67%
59	71	22	69%
60	815	275	66%

Tabela 2 – Comparação da busca cega e a primeira heurística

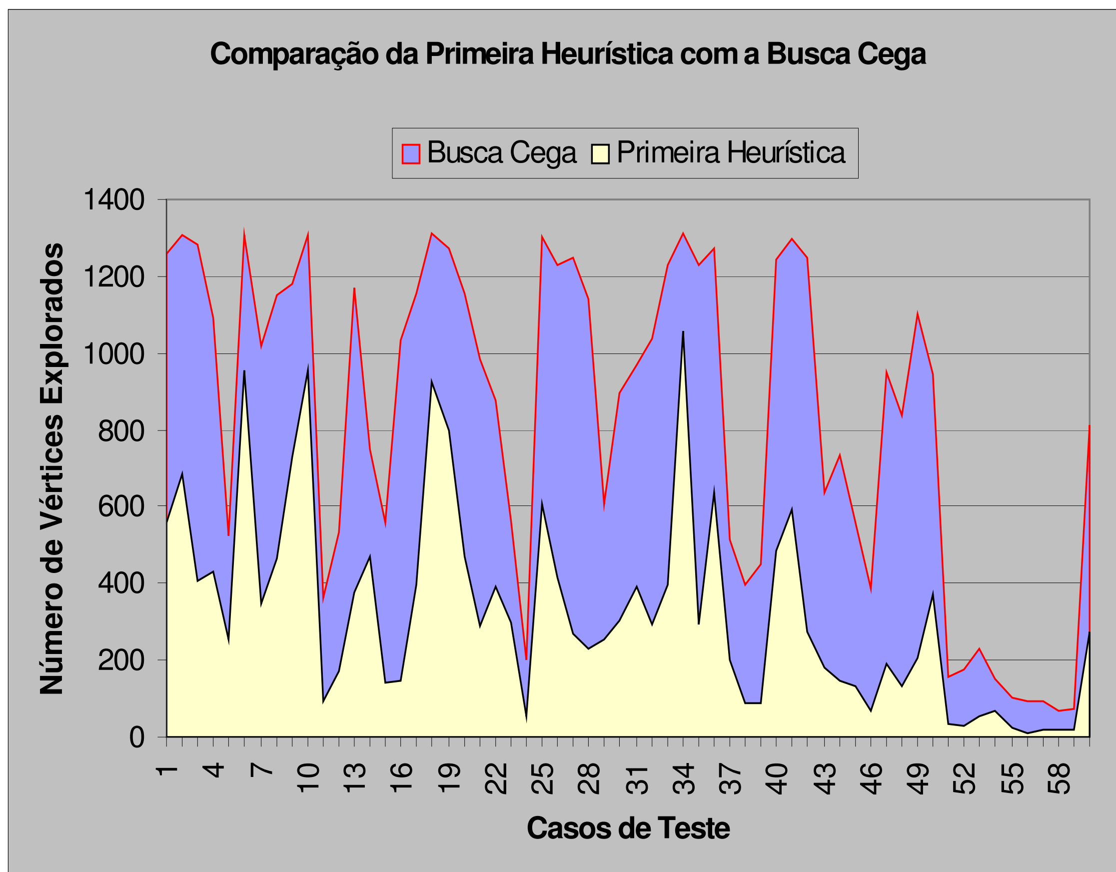


Figura 12 - Gráfico da comparação da primeira heurística com a busca cega

Com base nos resultados dos casos de teste apresentados, pode-se concluir que esta heurística é eficiente, uma vez que conseguiu reduzir o número de vértices explorados a uma média de 61%, e todos os caminhos obtidos usando-se esta heurística estavam corretos. Isso significa que o algoritmo não encerrou antes do tempo por causa da mudança de prioridade dos vértices.

4.6 A SEGUNDA HEURÍSTICA

Pode-se afirmar que a procura de heurísticas para o PMC nada mais é que uma tentativa de informar ao algoritmo quais vértices são os mais promissores e, por conseguinte, devem ter maior prioridade para serem explorados. Deste ponto do trabalho em diante, serão apresentadas algumas outras heurísticas, as quais serão avaliadas afim de classificá-las como eficientes ou não. Cada uma delas será testada individualmente no algoritmo.

Baseando-se na figura 13, pode-se afirmar que quando se deseja identificar o menor caminho para ir do vértice **A** até o vértice **B**, todos os vértices em preto serão explorados, pois estão mais perto da origem do que o vértice destino está. Por caminho entende-se o somatório das distâncias de todos os trechos de um dado percurso.

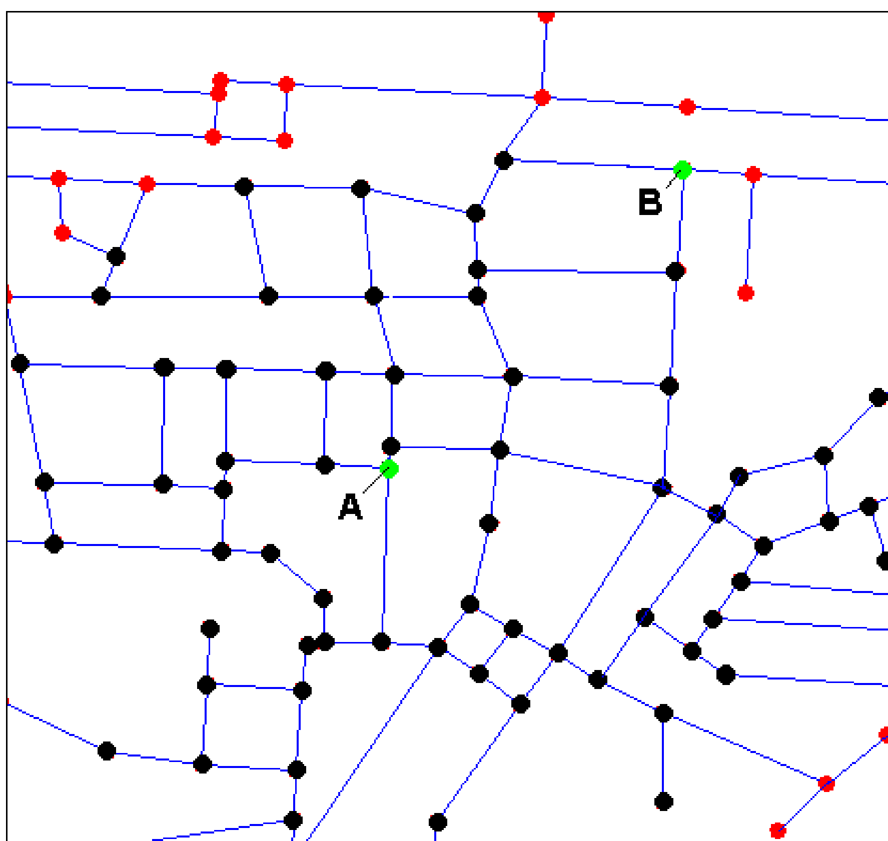


Figura 13 – Vértices explorados antes de encontrar o vértice destino

A heurística sugerida agora é baseada no princípio de que normalmente o menor caminho para se deslocar de um ponto à outro será formado por vértices que estejam entre os pontos origem e destino. Na verdade, este princípio é o mesmo da primeira heurística. Para ilustrar a idéia, foi usada a figura 14.

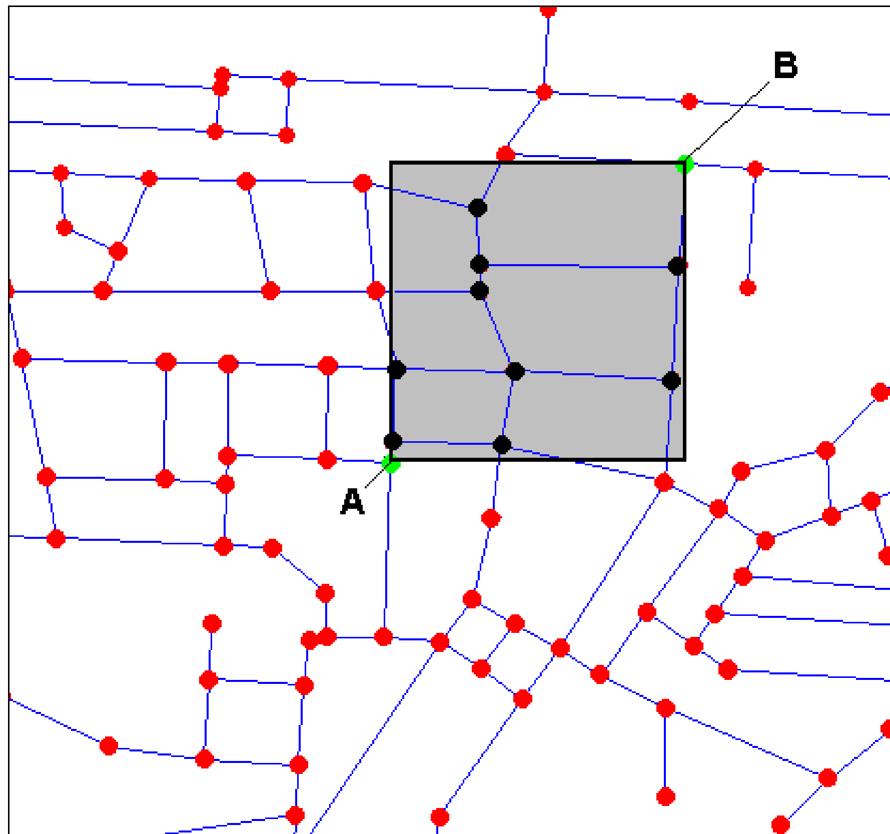


Figura 14 – Ilustração da segunda heurística

A idéia é dar maior prioridade para aqueles vértices que se encontram dentro do retângulo, cujos extremos são os vértices origem e destino. Pelo fato de a heurística ser apoiada no uso de um retângulo para delimitar os vértices que terão prioridade, doravante, esta heurística será chamada de Heurística do Retângulo.

É preciso denotar maior prioridade para os vértices dentro do retângulo de alguma maneira. Assim, aqueles vértices que se encontram fora da área delimitada pelo retângulo, receberam a **h’** igual a 0 (zero), para os vértices dentro do retângulo, foi atribuído um valor negativo dado em metros. Esta heurística não garante que o menor caminho seja escolhido, mas o erro será no máximo o valor atribuído à **h’**. Iniciou-se os testes com o **h’** com valor de

100m, depois com 500m, 1000m e 2000m. A tabela 3 mostra os resultados obtidos com os casos de testes.

N° Do TESTE	B. Cega	**h'** = 100m		**h'** = 500m		**h'** = 1000m		**h'** = 2000m	
	Vértices Explor.	Vértices Explor.	% Redução	Vértices Explor.	% Redução	Vértices Explor.	% Redução	Vértices Explor.	% Redução
1	1258	1254	0%	1202	4%	1089	13%	851	32%
2	1309	1307	0%	1307	0%	1307	0%	1307	0%
3	1283	1283	0%	1282	0%	1266	1%	1148	11%
4	1091	1065	2%	963	12%	929	15%	726	33%
5	524	500	5%	428	18%	366	30%	244	53%
6	1306	1306	0%	1303	0%	1303	0%	1249	4%
7	1019	1000	2%	931	9%	846	17%	711	30%
8	1150	1145	0%	1159	-1%	1179	-3%	1204	-5%
9	1180	1174	1%	1188	-1%	1208	-2%	1233	-4%
10	1307	1307	0%	1306	0%	1306	0%	1306	0%
11	364	354	3%	321	12%	321	12%	321	12%
12	534	518	3%	518	3%	518	3%	518	3%
13	1172	1159	1%	1168	0%	1180	-1%	1180	-1%
14	747	733	2%	700	6%	700	6%	700	6%
15	556	516	7%	370	33%	297	47%	221	60%
16	1031	1000	3%	933	10%	935	9%	935	9%
17	1156	1128	2%	1125	3%	1125	3%	1125	3%
18	1311	1311	0%	1311	0%	1311	0%	1311	0%
19	1271	1265	0%	1256	1%	1256	1%	1256	1%
20	1153	1131	2%	1046	9%	1053	9%	1043	10%
21	983	945	4%	934	5%	934	5%	934	5%
22	876	848	3%	810	8%	810	8%	810	8%
23	563	538	4%	495	12%	495	12%	495	12%
24	199	191	4%	147	26%	147	26%	147	26%
25	1301	1298	0%	1288	1%	1288	1%	1288	1%
26	1229	1226	0%	1226	0%	1226	0%	1226	0%
27	1246	1241	0%	1204	3%	1156	7%	1065	15%
28	1139	1123	1%	1123	1%	1123	1%	1123	1%
29	609	581	5%	504	17%	504	17%	504	17%
30	897	869	3%	806	10%	777	13%	720	20%
31	968	940	3%	886	8%	829	14%	786	19%
32	1039	1016	2%	975	6%	975	6%	975	6%
33	1228	1224	0%	1224	0%	1224	0%	1224	0%
34	1310	1309	0%	1299	1%	1299	1%	1299	1%
35	1230	1215	1%	1222	1%	1223	1%	1223	1%
36	1275	1269	0%	1269	0%	1269	0%	1269	0%
37	514	493	4%	482	6%	482	6%	482	6%
38	396	378	5%	377	5%	377	5%	377	5%
39	448	425	5%	380	15%	380	15%	380	15%
40	1244	1228	1%	1214	2%	1214	2%	1265	-2%
41	1298	1296	0%	1291	1%	1285	1%	1196	8%

42	1250	1239	1%	1157	7%	1184	5%	1184	5%
43	636	613	4%	552	13%	487	23%	393	38%
44	736	723	2%	723	2%	723	2%	723	2%
45	558	536	4%	461	17%	358	36%	145	74%
46	387	361	7%	265	32%	165	57%	74	81%
47	949	909	4%	762	20%	591	38%	395	58%
48	835	789	6%	793	5%	793	5%	793	5%
49	1100	1063	3%	1011	8%	1011	8%	1011	8%
50	944	928	2%	880	7%	880	7%	880	7%
51	159	141	11%	64	60%	30	81%	30	81%
52	176	156	11%	84	52%	59	66%	38	78%
53	228	196	14%	117	49%	60	74%	45	80%
54	152	136	11%	136	11%	136	11%	136	11%
55	102	79	23%	32	69%	23	77%	23	77%
56	94	83	12%	39	59%	24	74%	24	74%
57	92	79	14%	37	60%	25	73%	25	73%
58	67	58	13%	37	45%	34	49%	34	49%
59	71	63	11%	46	35%	43	39%	43	39%
60	815	786	4%	724	11%	649	20%	461	43%
Total			2%		6%		9%		12%

Tabela 3 – Comparação entre a busca cega e a Heurística do Retângulo

Com os testes aplicados, se pode perceber que esta heurística mostrou-se pouco eficiente a nível geral, pois com um valor de 2000m para o **h’**, obteve uma taxa média de redução do número de vértices explorados da ordem de 12%, contra 61% da primeira heurística. Além disso, à medida que cresce o **h’**, cresce também o número de caminhos que não são os mais curtos, como pode ser visto na tabela 4. Alguns casos de testes ainda, tiveram uma porcentagem negativa, ou seja, exploraram mais vértices do que usando a busca cega. A figura 15, mostra um gráfico com a comparação da porcentagem de redução do número de vértices explorados de acordo com o valor de **h’**.

Valor de **h’** Em metros	Caminhos errados
100	1
500	5
1000	10
2000	11

Tabela 4 – Caminhos errados em função do valor de h’

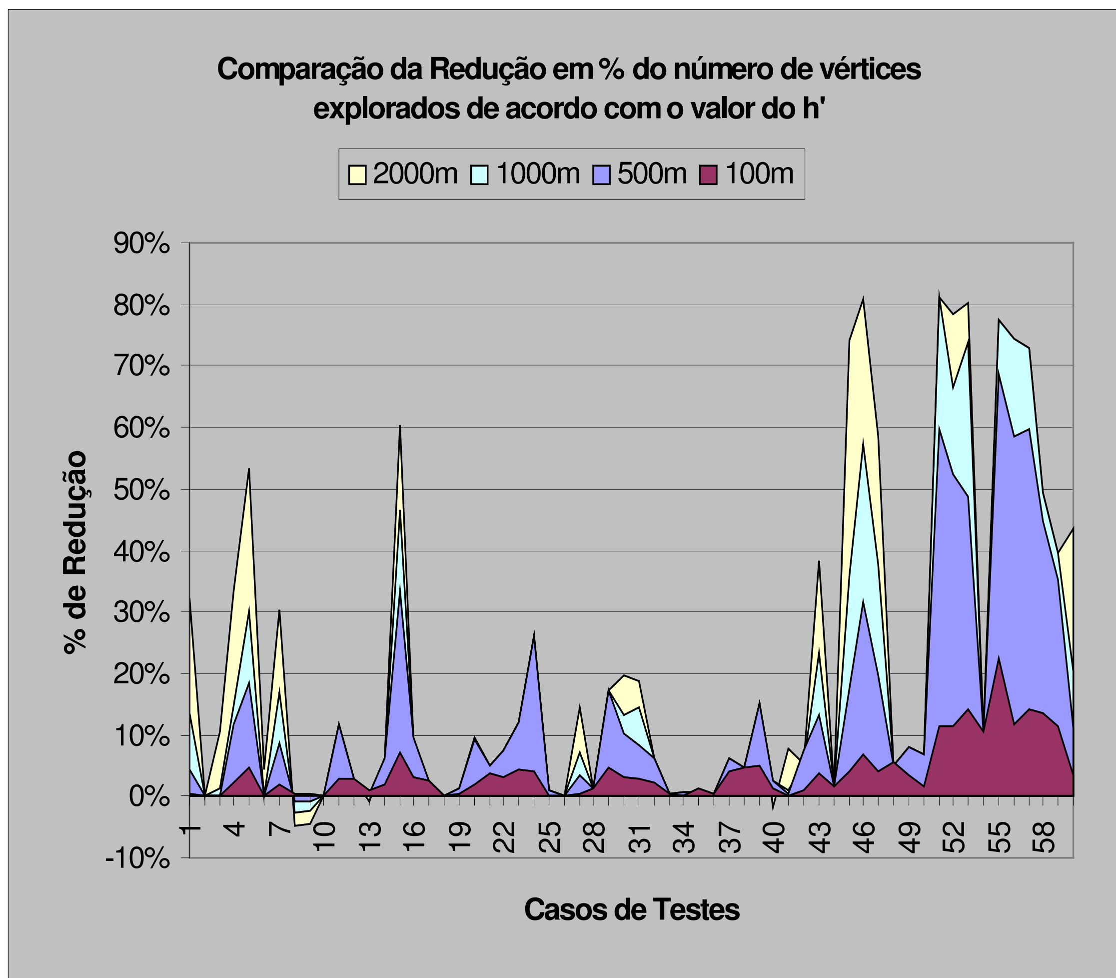


Figura 15 – Gráfico com a comparação da porcentagem de redução do número de vértices explorados de acordo com o valor de h’

A julgar pelos resultados genéricos obtidos com a Heurística do Retângulo, pode-se concluir que ela não é eficiente, pelo menos para esta base de testes, e não garante a melhor solução. Então como explicar o fato de que vários testes tiveram resultados até melhores do que os obtidos com a primeira heurística, passando de 80%, e traçaram o caminho corretamente ? Para responder a essa pergunta, é preciso analisar os casos de testes que se comportaram dessa maneira. Todos têm em comum o fato de o caminho ideal passar se não por completo, mas a sua maior parte por dentro do retângulo criado. Assim, conclui-se que esta heurística pode ser aplicada em bases de dados onde, não importando em qual posição seja criado o retângulo, sempre exista o caminho ideal dentro dele ou em suas proximidades. A figura 16 ilustra alguns dos testes bem sucedidos usando a Heurística do Retângulo. Cada

cor indica o retângulo traçado entre os vértices origem e destino, além de mostrar o caminho traçado.

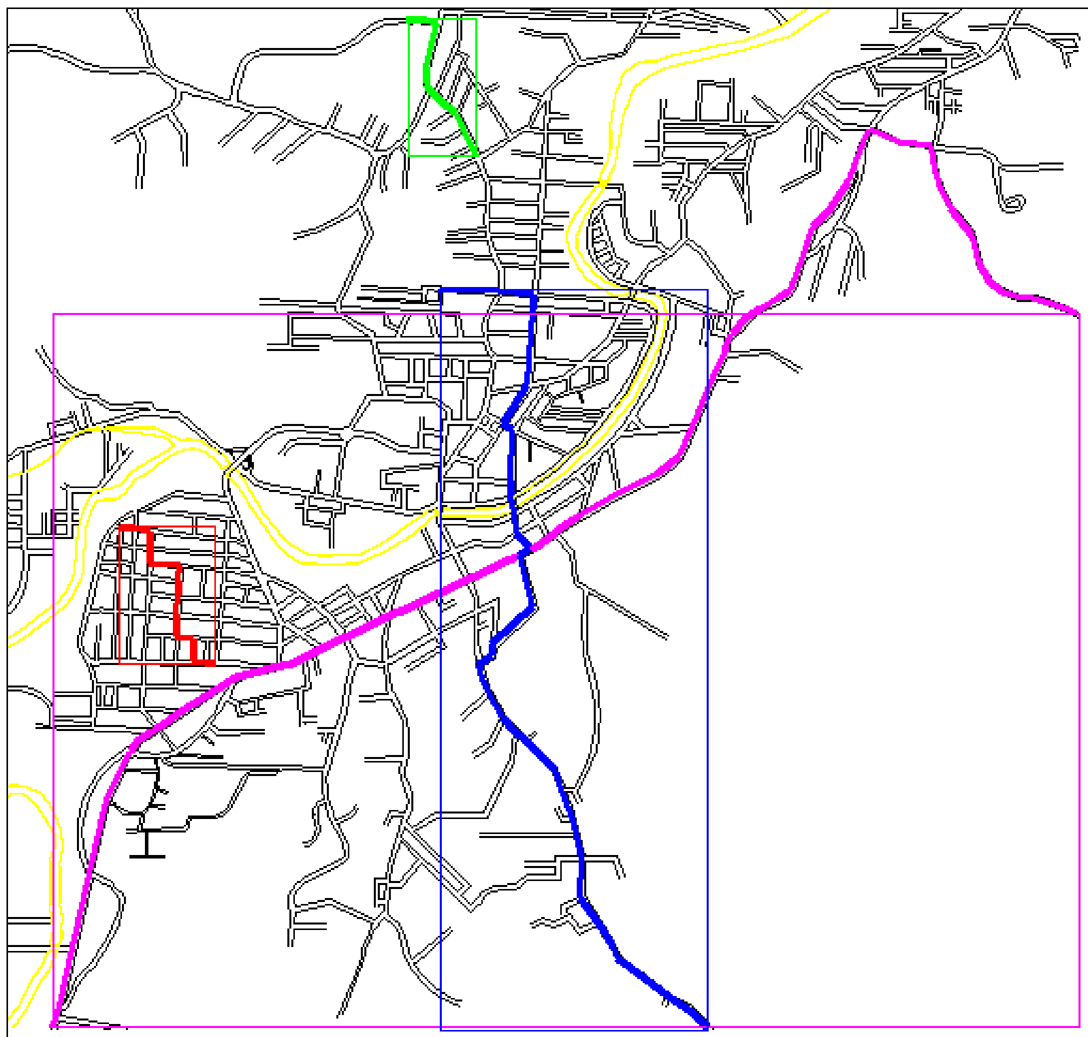


Figura 16 – Exemplos de casos de testes bem sucedidos com a Heurística do Retângulo

4.7 VÁRIOS VÉRTICES COMO DESTINO

O estudo desenvolvido até aqui foi para atender o primeiro objetivo do trabalho, ou seja, o estudo de heurísticas para melhorar o desempenho na busca da solução do PMC. Esta seção irá tratar do segundo objetivo: permitir a determinação do menor caminho tendo vários vértices como destino. Vale frisar que a ordem em que os vértices destinos devem ser visitados não é conhecida, pois se fosse, bastaria encontrar pelo estudado o menor caminho da

origem até o primeiro destino, depois do segundo destino até o terceiro e assim por diante, até completar todo o caminho que atendesse à todas as metas.

A fim de entender a dimensão do problema em se definir um caminho que passa por vários vértices pré determinados, é preciso entender a estrutura do algoritmo. Este explora os vértices do grafo começando pelo vértice origem. Os vértices de menor custo são explorados primeiro e, a cada passo do algoritmo, um novo vértice é explorado e todos os seus vértices adjacentes são adicionados na fila *abertos*, se já não foram. Este processo se repete até que o algoritmo escolha o vértice destino para explorar. O processamento pára e o menor caminho entre a origem e o destino é definido ligando-se alguns vértices que estão na fila *fechados* através da indicação do vértice pai contida em cada um deles.

Analisando-se o algoritmo superficialmente, pode-se deixar levar pela impressão de que para resolver o problema com vários destinos, bastaria parar o algoritmo somente quando ele tivesse explorado todos os vértices destinos. Isso não é verdade, pelo simples fato que o algoritmo pode mudar o caminho de acordo com o custo de cada vértice. Não há garantias de que mesmo tendo explorado todos os vértices destino, o caminho encontrado passe por todos eles.

Como o algoritmo não oferece condições de fixar alguns vértices no caminho encontrado, pode-se concluir que da maneira que ele foi concebido, não pode ser aplicado para resolver o problema do menor caminho com vários vértices de destino. Diante desta constatação, faz-se necessário encontrar outras formas para resolver este problema. Esta seção sugere uma saída através de heurísticas.

4.7.1 RESOLVENDO O PROBLEMA

Assim como podem existir vários caminhos possíveis quando existe um destino, o mesmo acontece para vários destinos. Considerando a figura 17, se o problema for definir o caminho para sair do vértice **A** e alcançar os vértices **B**, **C** e **D**, não importando a ordem, mas o somatório das distâncias sendo o menor possível, pode-se perceber que não é muito fácil resolvê-lo, pois existem vários caminhos possíveis desconsiderando as distâncias, mas somente um, ou poucos, na verdade resolvem o problema. Então como definir qual é o caminho correto?

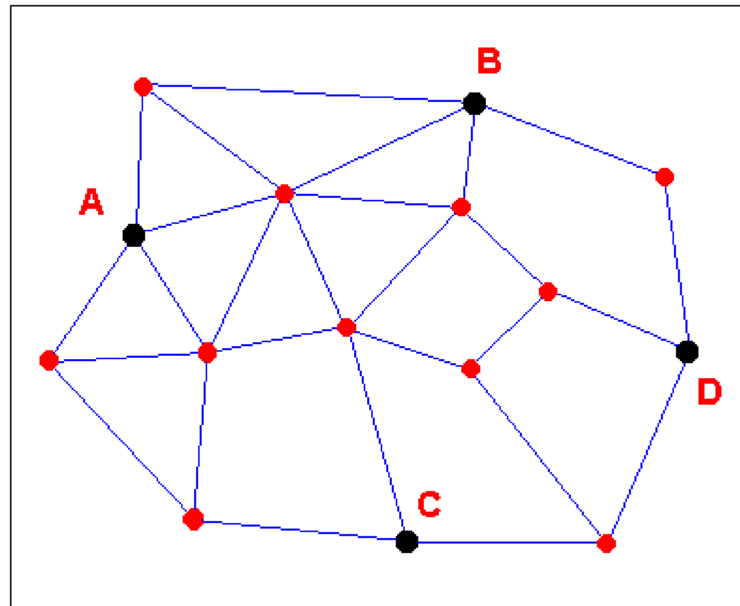


Figura 17 – Grafo ilustrando o PMC com vários destinos

Para saber qual o menor caminho possível, primeiro é preciso saber quais são os caminho possíveis, ou seja, em quais ordem os destinos podem ser atendidos. Tendo-se definido a ordem para passar por todos os destinos, determinar o menor caminho entre cada um deles fica fácil, bastando usar o algoritmo para isso. Neste exemplo, a ordem de visita dos destinos poderia ser uma das seguintes:

a) A → B → C → D;

b) A → B → D → C;

c) A → C → B → D;

d) A → C → D → B;

e) A → D → C → B;

f) A → D → B → C.

Pode-se concluir que o número de caminhos possíveis é o resultado do número de combinações possíveis entre os destinos. A única maneira de saber qual desses possíveis caminhos representa o mais curto, é sabendo o comprimento de cada um deles. Isto, em muitos casos, se torna impraticável, pois para determinar o comprimento de cada um dos possíveis caminhos, é preciso, como já foi dito, aplicar o algoritmo e encontrar o menor

caminho entre cada um dos vértices integrantes do caminho, na ordem em que eles são apresentados. As heurísticas ajudariam, mas mesmo assim, o tempo para a resolução seria grande. Conclui-se assim, que na prática, este método não seria viável.

Como o problema consiste em determinar a ordem em que os destinos serão visitados, a proposta de solução que será apresentada baseia-se em defini-la através de heurística. Já que as heurísticas não garantem o melhor resultado, mas sim um resultado válido obtido de maneira mais rápida, então este método não poderá ser aplicado quando for necessário saber com precisão qual é o menor caminho.

A heurística proposta é definir a ordem de visita dos vértices destinos da mesma forma como mostrado anteriormente, ou seja, definindo-se primeiramente os possíveis caminhos, mas ao invés de usar o algoritmo para definir o menor caminho entre cada um dos vértices destinos, usar a distância geográfica. Depois de definida a ordem, basta calcular o menor caminho real entre cada um dos vértices. O princípio da heurística é baseado no fato de que na maioria dos casos de testes apresentados, o caminho encontrado passava bem próximo de uma linha imaginária ligando os dois pontos.

5 O SISTEMA DESENVOLVIDO

Com o propósito de implementar e testar o algoritmo e as heurísticas apresentadas nos capítulos anteriores, foi criado um protótipo de um sistema para encontrar o menor caminho em um mapa de uma cidade especificada.

5.1 AMBIENTE DE DESENVOLVIMENTO

A base de dados foi construída usando-se o Microsoft Access. A implementação foi construída em Java™, linguagem de programação da Sun Microsystems Inc. O ambiente de desenvolvimento foi o Kawa versão 3.0 da Webcetera Inc. Para a construção da base de testes foram usados alguns *softwares* para conversão e construção dos dados, como o AutoCAD R13, da AutoDesk Inc. e o Microsoft Excel.

5.2 COMPONENTES DO SISTEMA

Serão mostrados agora, os principais componentes do sistema, tanto os componentes internos, como a base de dados, quanto os externos, como a interface com o usuário.

5.2.1 A BASE DE DADOS

Serão apresentados agora os arquivos que foram criados para armazenar as informações do sistema. Deve-se frisar que o protótipo não foi concebido para gerenciar os dados contidos na base de dados, portanto as suas funções estão limitadas a usar o dados existentes.

A estrutura de adjacência foi a escolhida para comportar os dados relativos ao grafo gerado a partir da base de testes. Para o algoritmo em si, sem usar heurísticas, bastaria ter uma tabela contendo a estrutura de adjacência do grafo, mas como o fator distância geográfico foi usado nas heurísticas, como explicado no capítulo 4, foi necessário criar também uma tabela para guardar as coordenadas geográficas de cada vértice do grafo. Porém, se somente essas duas tabelas fossem criadas, o cálculo do menor caminho seria possível, mas toda a interação com o usuário não, esse é o motivo de várias tabelas terem sido criadas, como é mostrado na figura 18 através do Modelo Entidade Relacionamento – MER.

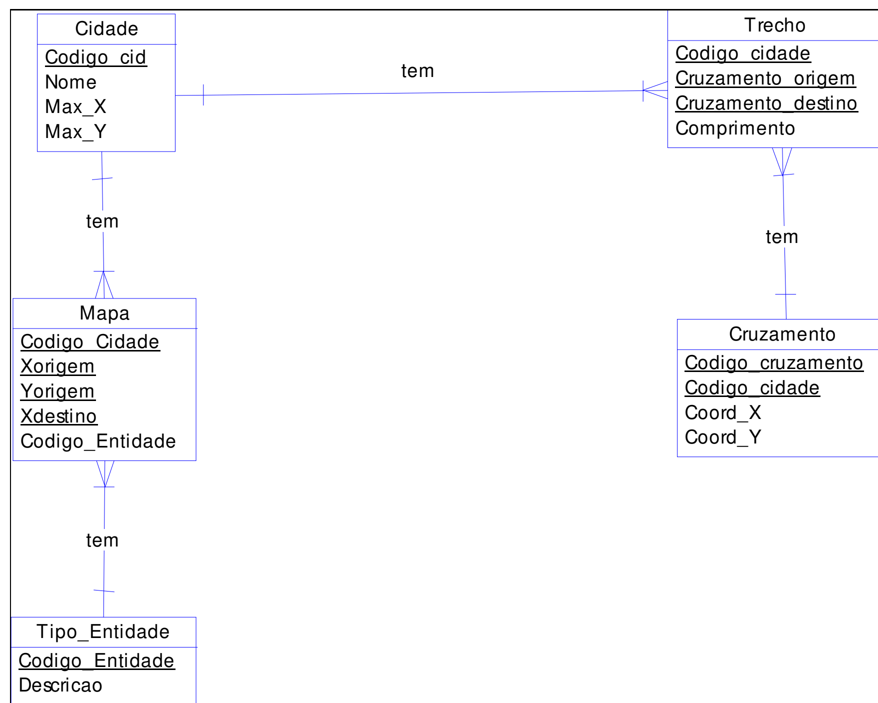


Figura 18– MER da base de dados do sistema

Os eventos principais do protótipo são os seguintes:

a) usuário escolhe cidade;

b) usuário solicita caminho.

Esses eventos podem ser vistos no Diagrama de Contexto e também no Diagrama de Fluxo de Dados, respectivamente as figuras 19 e 20:

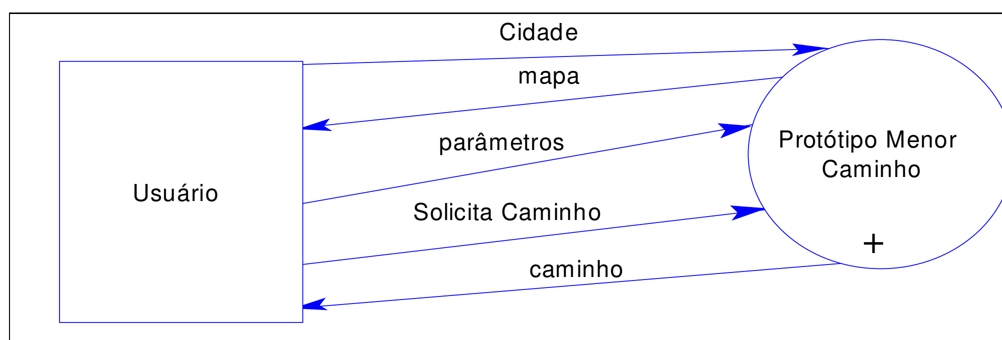


Figura 19 – Diagrama de Contexto

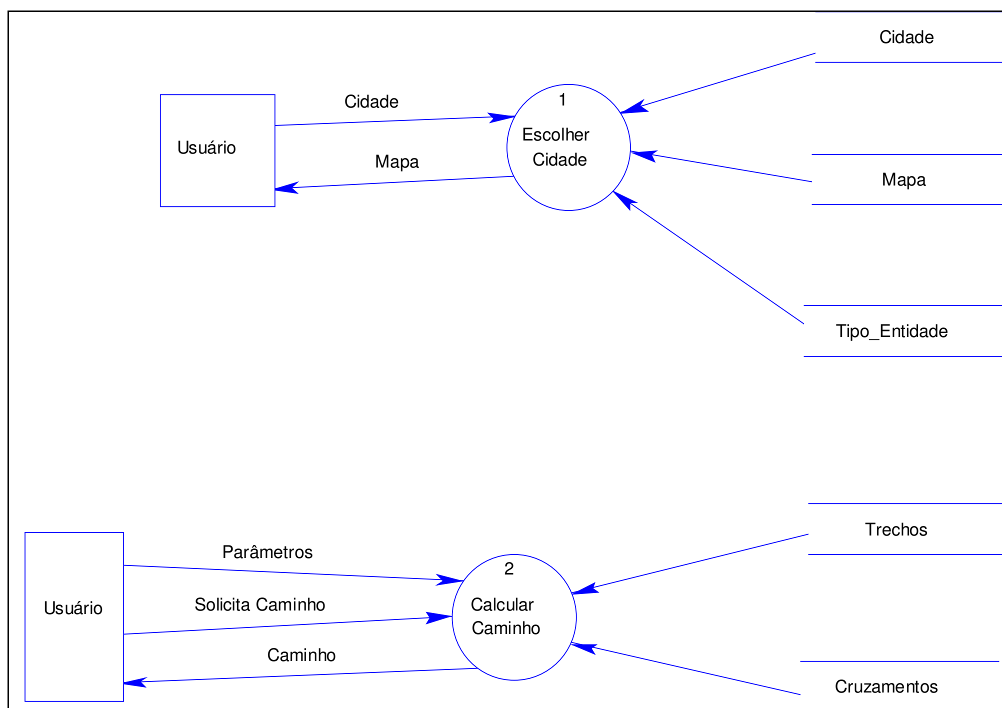


Figura 20 – Diagrama de Fluxo de Dados

5.2.1.1 DICIONÁRIO DE DADOS

A seguir as tabelas usadas no protótipo serão explicadas, mostrando-se os seus campos e uso. Os campos em negrito representam os campos que formam a chave primária da tabela:

TRECHO – Contém a estrutura de adjacência. É chamada de TRECHOS, pois as arestas representam, em sua maioria, trechos de logradouros e não eles por completo.

CAMPOS	DESCRIÇÃO
Cruzamento_origem	Código do vértice inicial da aresta.
Cruzamento_destino	Código do vértice final da aresta.
Codigo_cidade	Código da cidade a que pertence a aresta.
Comprimento	Comprimento em metros da aresta.

CRUZAMENTO – Contém os vértices usados na estrutura de adjacência. Guarda as coordenadas do vértice para poder usá-las nas heurísticas e desenhar o grafo na tela.

CAMPOS	DESCRIÇÃO
Codigo_cruzamento	Código do vértice.
Codigo_cidade	Código da cidade a que pertence o vértice.
Coord_X	Coordenada do eixo X em metros.
Coord_Y	Coordenada do eixo Y em metros.

CIDADE – Contém informações das cidades.

CAMPOS	DESCRIÇÃO
Codigo_cid	Código da cidade.
Nome	Nome da cidade.
Max_X	Extensão em metros da cidade no eixo X. Esse valor serve para limitar a área de visualização da cidade.
Max_Y	Extensão em metros da cidade no eixo Y. Esse valor serve para limitar a área de visualização da cidade.

MAPA – Contém informações dos mapas das cidades, como desenho das ruas, rios, etc. Serve apenas para facilitar o entendimento e localização do usuário. Essa tabela, na verdade, guarda apenas os pontos extremos de linhas e o que estas linhas representam.

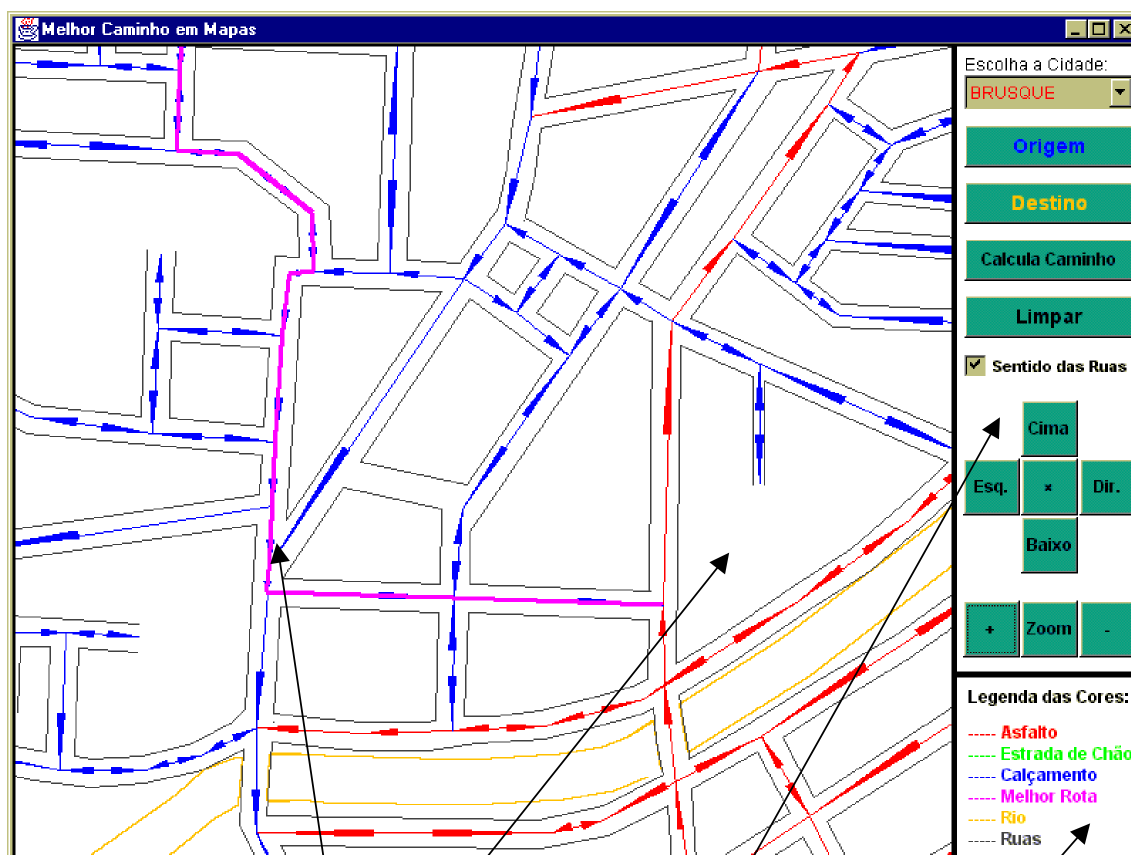
CAMPOS	DESCRIÇÃO
Codigo_cidade	Código da cidade.
Xorigem	Coordenada do eixo X da origem da linha representada.
Yorigem	Coordenada do eixo Y da origem da linha representada.
Xdestino	Coordenada do eixo X do destino da linha representada.
Ydestino	Coordenada do eixo Y do destino da linha representada.
Codigo_Entidade	Código do tipo da linha, como rio, rua, etc.

TIPO_ENTIDADE – Contém os tipos de entidades que são guardadas na tabela MAPAS. Esse tipo é usado para diferenciar, através de cores por exemplo, as diferentes entidades possíveis mostradas no mapa, como estradas, rios, mar, prédios, etc.

CAMPOS	DESCRIÇÃO
Codigo_entidade	Código do tipo de entidade.
Descricao	Nome que identifica o tipo de entidade.

5.2.2 AS TELAS

O sistema é composto por apenas uma tela, que é dividida em três áreas: uma para visualização do mapa, outra para controle e a outra serve apenas para mostrar uma legenda dos elementos dispostos na área de visualização do mapa. A figura 21 ilustra a tela do protótipo.



Visualização do mapa Controles Legenda

Figura 21 – Ilustração da interface do protótipo

5.3 OPERAÇÃO DO SISTEMA

A função básica do protótipo é possibilitar o cálculo do melhor caminho para testes de performance das heurísticas estudadas anteriormente. Porém, algumas funções foram introduzidas para facilitar a interação com o usuário e, conseqüentemente, facilitar os testes efetuados. Dentre estas facilidades, pode-se citar o *zoom*, que permite ampliar alguma área específica do mapa, a identificação do sentidos das ruas, que ajuda a verificar se o caminho encontrado está correto, e os botões de navegação, que deslocam a posição do mapa.

5.3.1 ENCONTRANDO O MENOR CAMINHO

Agora serão mostrados os passos para encontrar o menor caminho entre dois vértices definidos dentro de uma cidade escolhida:

a) **escolher a cidade**: inicialmente, deve-se escolher com qual cidade se irá trabalhar. Para fazer isto, basta clicar na caixa de seleção identificada por "Escolha a Cidade" no canto superior direito e escolher a cidade desejada. Quando esta for escolhida, o seu respectivo mapa será desenhado na área de visualização do mapa.

b) **escolher o vértice de origem**: tendo escolhido a cidade, todos os seus dados foram carregados para a memória, agora pode-se escolher qual será o vértice de origem. Para isto, clique no botão "Origem". Quando este botão é acionado, todos os vértices do grafo que representa o mapa são desenhados na área de visualização do mapa. O usuário deve agora, escolher qual deles deseja selecionar, clicando com o mouse sobre ele. Assim que o vértice foi escolhido, todos desaparecem da tela e pode-se passar para o próximo passo.

c) **escolher o vértice de destino**: o princípio do funcionamento é o mesmo usado para definir a origem, a diferença é que deve-se apertar o botão "Destino".

d) **ordenar o cálculo do menor caminho**: como já foram selecionados os vértices de origem e destino, o algoritmo já tem informações suficientes para poder encontrar o menor caminho. Para acionar o algoritmo, basta clicar sobre o botão "Calcula Caminho" e aguardar alguns instantes. Enquanto o computador está processando os

cálculos, aparece uma ampulheta, que some assim que o menor caminho for encontrado. Ele é indicado no mapa, traçando-se uma linha cor-de-rosa que indica a rota que deve ser tomada. Se o usuário não quiser mais ver o traçado da rota, ele pressiona o botão “Limpar”. Todo o processo pode ser repetido.

5.3.2 RECURSOS DE NAVEGAÇÃO

Para facilitar a visualização do mapa, alguns recursos de navegação pelo mapa foram introduzidos. Eles estão divididos em duas categorias: *zoom* e deslocamento.

Os recursos de *zoom* são fornecidos através de três botões, conforme a figura 22:

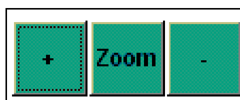


Figura 22 – Botões de *zoom*

O botão da esquerda tem a função de aumentar a área do mapa que está sendo visualiza atualmente. O da direita diminui esta mesma área. Com o botão central, pode-se determinar qual área do mapa será ampliada. Para isso, basta indicar um canto da área clicando com o botão esquerdo do mouse e, com o botão ainda pressionado, deslocar o cursor do mouse até o segundo extremo da área desejada e soltar o botão. O protótipo pegará o lado maior da área escolhida e ampliará o mapa até ocupar toda a área disponível.

Para os recursos de deslocamento, existem cinco botões, conforem a figura 23:

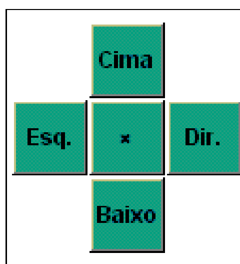


Figura 23 – Botões de deslocamento

Os quatro botões das extremidades deslocam o campo de visão do usuário de acordo com as direções indicadas nos botões. O botão do meio, restaura tanto a posição quanto a

escala inicial do mapa, ou seja, todo o mapa da cidade é visualizado de forma centralizada na tela.

5.3.3 IDENTIFICANDO O SENTIDO DAS RUAS

Quando o usuário quiser saber qual o sentido das ruas do mapa, bastará acionar sobre a *checkbox* (caixa de checagem) identificada por "Sentido das Ruas". São desenhadas setas no meio dos trechos das ruas. Se existir apenas uma seta, o trecho tem mão única, caso duas setas sejam desenhadas, o trecho tem mão dupla. Note que o tamanho das setas não varia de acordo com o *zoom* do mapa, portanto, em alguns casos mostrar o sentido das ruas pode afetar a visualização e entendimento do mapa.

5.4 PRINCIPAIS PROGRAMAS FONTES

Nesta seção se encontram os principais códigos fontes implementados.

5.4.1 BUSCA CEGA

O quadro 4 traz o código fonte da implementação do algoritmo usando a busca cega.

```
public void calcula_caminho(int origem, int destino)
{
  // Zera as listas
  abertos.removeAllElements();
  fechados.removeAllElements();
  caminho.removeAllElements();

  vAtual = new Vertice(origem, 0, 0, 0);
  abertos.addElement(vAtual);
  boolean explorado;

  // Enquanto a lista de abertos possui algum elemento
  while (!(abertos.isEmpty()))
  { // retira o vértice de menor f' da lista de abertos
    vAtual = exclui_abertos();
    // inclui o vértice na lista de fechados
    fechados.addElement(vAtual);

    // Se o vértice atual não é o destino
    if (vAtual.getCodigo() != destino)
    {
      int tamTrechos = vm.Trechos.size();
      Trecho t = new Trecho(0,0,0,0,0,0);

      // Abre todos os vértices adjacentes ao vértice atual
      for (int i=0; i < tamTrechos; i++)
      { t = (Trecho)vm.Trechos.elementAt(i);

        // Se o trecho que está sendo avaliado é adjacente ao atual
        if (t.getOrigem() == vAtual.getCodigo())
        {
```

```
          Vertice vFilho = new Vertice(0,0,0,0);
          vFilho.setCodigo(t.getDestino());
          vFilho.setPai(t.getOrigem());
        vFilho.setCusto(t.getComprimento() + vAtual.getCusto());
        explorado = false;
          int ifechados = 0;

          // Procura se o vértice já foi explorado  na lista de fechados
          while (ifechados < fechados.size())
          {
            vTmp = (Vertice)fechados.elementAt(ifechados);
            if (vFilho.getCodigo() == vTmp.getCodigo())
            {
              explorado = true;
              ifechados = fechados.size();
            }
            ifechados++;
          }

          if (!(explorado))
          {
            int iabertos = 0;

            // Varre toda a lista de abertos
            while (iabertos < abertos.size())
            {
              vTmp = (Vertice)abertos.elementAt(iabertos);
              if (vFilho.Codigo == vTmp.Codigo)
              {
                explorado = true;
                iabertos = abertos.size();

                // Se o vértice sendo expl. atualmente tiver custo menor
                // que o que está na fila abertos, então deve-se trocá-los
                if (vFilho.getCusto() < vTmp.getCusto())
                {
                  vFilho.setHeuristica(vTmp.getHeuristica());
                  abertos.removeElement(vTmp);
                  abertos.addElement(vFilho);
                }
              }
              iabertos++;
            }
          }

        if (!(explorado))   abertos.addElement(vFilho);

        }
      }
    }

    else
    { // Encontrou o melhor caminho que é armazenado no vetor "caminho"

      caminho.addElement(vAtual);
      int ifechados = fechados.size();
      int anterior = vAtual.getPai();

      vTmp = new Vertice(0,0,0,0);

      while (ifechados > 0)
      {
        vTmp = (Vertice)fechados.elementAt(ifechados-1);

        if (vTmp.getCodigo() == anterior)
        {
```

```
            anterior = vTmp.getPai();
            caminho.addElement(vTmp);
          }
          ifechados--;
        }

        abertos.removeAllElements(); // Para parar o while
      }
    }
}
```

Quadro 4 – Código fonte do algoritmo usando a busca cega

5.4.2 A PRIMEIRA HEURÍSTICA

O quadro 5 traz o código fonte da implementação do algoritmo usando a primeira heurística.

```
public void calcula_caminho(int origem, int destino)
{
  int xDestino, yDestino, xAtual, yAtual;

  // Pega as coordenadas do vértice destino
  c = (Cruzamento)vm.Cruzamentos.elementAt(destino - 1);
  xDestino = (int)(c.getCoord_X());
  yDestino = (int)(c.getCoord_Y());

  // Zera as listas
  abertos.removeAllElements();
  fechados.removeAllElements();
  caminho.removeAllElements();

  vAtual = new Vertice(origem, 0, 0, 0);
  abertos.addElement(vAtual);
  boolean explorado;

  // Enquanto a lista de abertos possui algum elemento
  while (!(abertos.isEmpty()))
  {
    // retira o vértice de menor f’ da lista de abertos
    vAtual = exclui_abertos();
    // inclui o vértice na lista de fechados
    fechados.addElement(vAtual);

    // Se o vértice atual não é o destino
    if (vAtual.getCodigo() != destino)
    {
      int tamTrechos = vm.Trechos.size();
      Trecho t = new Trecho(0,0,0,0,0,0);

      // Abre todos os vértices adjacentes ao vértice atual
      for (int i=0; i < tamTrechos; i++)
      {
        t = (Trecho)vm.Trechos.elementAt(i);
```

```
        // Se o trecho que está sendo avaliado é adjacente ao atual
        if (t.getOrigem() == vAtual.getCodigo())
        { Vertice vFilho = new Vertice(0,0,0,0);
          vFilho.setCodigo(t.getDestino());
          vFilho.setPai(t.getOrigem());
          vFilho.setCusto(t.getComprimento() + vAtual.getCusto());
          explorado = false;
          int ifechados = 0;

          // Procura se o vértice já foi explorado  na lista de fechados
          while (ifechados < fechados.size())
          { vTmp = (Vertice)fechados.elementAt(ifechados);
            if (vFilho.getCodigo() == vTmp.getCodigo())
            { explorado = true;
              ifechados = fechados.size();
            }
            ifechados++;
          }

          if (!(explorado))
          { int iabertos = 0;

            // Varre toda a lista de abertos
            while (iabertos < abertos.size())
            { vTmp = (Vertice)abertos.elementAt(iabertos);
              if (vFilho.Codigo == vTmp.Codigo)
              {
                explorado = true;
                iabertos = abertos.size();

                // Se o vértice sendo expl. atualmente tiver custo menor
                // que o que está na fila abertos, então deve-se trocá-los
                if (vFilho.getCusto() < vTmp.getCusto())
                {
                  vFilho.setHeuristica(vTmp.getHeuristica());
                  abertos.removeElement(vTmp);
                  abertos.addElement(vFilho);
                }
              }
              iabertos++;
            }
          }

        if (!(explorado))
        { // Pega as coordenadas do vértice atual
          c = (Cruzamento)vm.Cruzamentos.elementAt(vFilho.getCodigo() - 1);
          xAtual = (int)(c.getCoord_X());
          yAtual = (int)(c.getCoord_Y());

          // Calcula a distância entre o vértice atual e o destino
          long dist;
          dist = (long)Math.sqrt(Math.pow((yDestino-yAtual),2) +
                          Math.pow((xDestino-xAtual),2));
          vFilho.setHeuristica(dist);

          abertos.addElement(vFilho);
        }
      }
    }
  }

  else
  {
```

```
        // Encontrou o melhor caminho que agora é armazenado na vetor "caminho"

        caminho.addElement(vAtual);
        int ifechados = fechados.size();
        int anterior = vAtual.getPai();

        vTmp = new Vertice(0,0,0,0);

        while (ifechados > 0)
        {
          vTmp = (Vertice)fechados.elementAt(ifechados-1);

          if (vTmp.getCodigo() == anterior)
          {
            anterior = vTmp.getPai();
            caminho.addElement(vTmp);
          }
          ifechados--;
        }

        abertos.removeAllElements(); // Para parar o while
      }
    }

  }
```

Quadro 5 – Código fonte do algoritmo usando a primeira heurística

5.4.3 HEURÍSTICA DO RETÂNGULO

O quadro 6 traz o código fonte da implementação do algoritmo usando a Heurística do Retângulo.

```
  public void calcula_caminho(int origem, int destino)
  {
    int xOrigem, yOrigem, xDestino, yDestino, xAtual, yAtual;

    // Pega as coordenadas do vértice destino
    c = (Cruzamento)vm.Cruzamentos.elementAt(origem - 1);
    xOrigem = (int)(c.getCoord_X());
    yOrigem = (int)(c.getCoord_Y());

    // Pega as coordenadas do vértice destino
    c = (Cruzamento)vm.Cruzamentos.elementAt(destino - 1);
    xDestino = (int)(c.getCoord_X());
    yDestino = (int)(c.getCoord_Y());

    // Zera as listas
    abertos.removeAllElements();
    fechados.removeAllElements();
    caminho.removeAllElements();

    vAtual = new Vertice(origem, 0, 0, 0);
    abertos.addElement(vAtual);
    boolean explorado;

    // Enquanto a lista de abertos possui algum elemento
    while (!(abertos.isEmpty()))
    { // retira o vértice de menor f' da lista de abertos
      vAtual = exclui_abertos();
      // inclui o vértice na lista de fechados
      fechados.addElement(vAtual);
```

```
      // Se o vértice atual não é o destino
      if (vAtual.getCodigo() != destino)
      {
        int tamTrechos = vm.Trechos.size();
        Trecho t = new Trecho(0,0,0,0,0,0);

        // Abre todos os vértices adjacentes ao vértice atual
        for (int i=0; i < tamTrechos; i++)
        { t = (Trecho)vm.Trechos.elementAt(i);

          // Se o trecho que está sendo avaliado é adjacente ao atual
          if (t.getOrigem() == vAtual.getCodigo())
          {
            Vertice vFilho = new Vertice(0,0,0,0);
            vFilho.setCodigo(t.getDestino());
            vFilho.setPai(t.getOrigem());
            vFilho.setCusto(t.getComprimento() + vAtual.getCusto());
            explorado = false;
            int ifechados = 0;

            // Procura se o vértice já foi explorado  na lista de fechados
            while (ifechados < fechados.size())
            { vTmp = (Vertice)fechados.elementAt(ifechados);
              if (vFilho.getCodigo() == vTmp.getCodigo())
              { explorado = true;
                ifechados = fechados.size();
              }
              ifechados++;
            }

            if (!(explorado))
            {
              int iabertos = 0;
              // Varre toda a lista de abertos
              while (iabertos < abertos.size())
              { vTmp = (Vertice)abertos.elementAt(iabertos);
                if (vFilho.Codigo == vTmp.Codigo)
                { explorado = true;
                  iabertos = abertos.size();

                  // Se o vértice sendo expl. atualmente tiver custo menor
                  // que o que está na fila abertos, então deve-se trocá-los
                  if (vFilho.getCusto() < vTmp.getCusto())
                  {
                    vFilho.setHeuristica(vTmp.getHeuristica());
                    abertos.removeElement(vTmp);
                    abertos.addElement(vFilho);
                  }
                }
                iabertos++;
              }
            }

          if (!(explorado))
          {
            // Pega as coordenadas do vértice atual
            c = (Cruzamento)vm.Cruzamentos.elementAt(vFilho.getCodigo() - 1);
            xAtual = (int)(c.getCoord_X());
            yAtual = (int)(c.getCoord_Y());

            // Se o vértice estiver dentro do retângulo
            if ((((xAtual >= xOrigem) && (xAtual <= xDestino)) | ((xAtual <=
              xOrigem) && (xAtual >= xDestino))) && (((yAtual >= yOrigem) &&
              (yAtual <= yDestino)) | ((yAtual <= yOrigem) && (yAtual >=
              yDestino))))
            {
```

```
                    vFilho.setHeuristica((long)(vFilho.getHeuristica() - 2000));
                }

                abertos.addElement(vFilho);
              }
            }
          }
        }

        else
        { // Encontrou o melhor caminho que agora é armazenado na vetor "caminho"

          caminho.addElement(vAtual);
          int ifechados = fechados.size();
          int anterior = vAtual.getPai();

          vTmp = new Vertice(0,0,0,0);

          while (ifechados > 0)
          {
            vTmp = (Vertice)fechados.elementAt(ifechados-1);

            if (vTmp.getCodigo() == anterior)
            {
              anterior = vTmp.getPai();
              caminho.addElement(vTmp);
            }
            ifechados--;
          }
          abertos.removeAllElements(); // Para parar o while
        }
      }

    }
```

Quadro 6 – Código do fonte do algoritmo usando a Heurística do Retângulo

6 CONCLUSÃO

Este capítulo finaliza este trabalho mostrando uma avaliação dos resultados obtidos e fornece algumas sugestões para futuros trabalhos sobre o tema abordado.

6.1 CONSIDERAÇÕES FINAIS

Com o estudo realizado sobre a Teoria dos Grafos, nota-se que o grafo é uma ferramenta simples, mas ao mesmo tempo poderosa, capaz de representar e resolver muitos problemas da computação relacionados com a pesquisa.

Já com o estudo e implementação das heurísticas apresentadas, ficou claro que estas técnicas são bastante válidas e aplicam-se muito bem à resolução do PMC, tornando possível, como foi comprovado, o desenvolvimento de um protótipo capaz de suportar o uso de mapas de cidades com centenas ou milhares de cruzamentos, como se pretendia fazer. A Heurística do Retângulo, porém, não pode ser usada de maneira eficiente em qualquer base de dados, portanto, deve-se levar em consideração as limitações descritas no trabalho.

De acordo com o estudo das implicações do uso do algoritmo A* para resolver o PMC com vários destinos, conclui-se que este é um problema complexo e merecedor de um trabalho dedicado exclusivamente a ele.

6.2 EXTENSÕES

Durante a construção do protótipo, mais especificamente da base de dados, observou-se uma dificuldade em gerenciar as informações que compõem o sistema. Em virtude de mudanças constantes e repentinas que acontecem na malha viária das cidades, seria de grande valia um estudo e construção de um sistema, com interface amigável e intuitiva, capaz de gerenciar os dados dos mapas para tornar a atualização destes uma tarefa fácil, possibilitando assim, o uso do sistema apresentado aqui em situações reais.

Novas heurística poderiam ser agregadas ao algoritmo, o que abre a possibilidade de ser feito um novo estudo sobre elas, podendo inclusive, ser testado o algoritmo usando mais de uma heurística ao mesmo tempo, para verificar a eficiência destas associações. Seria relevante também, implementar o algoritmo e heurísticas mostrados aqui, utilizando outras

bases de dados, de preferência maiores, a fim de comparar os resultados obtidos com esse estudo.

Uma outra sugestão é fazer um estudo mais específico sobre o PMC com vários vértices como destino, implementando os métodos demonstrados e criando outros, se possível.

REFERÊNCIAS BIBLIOGRÁFICAS

[BOA79] BOAVENTURA NETTO, Paulo Osvaldo. **Teoria e modelos de grafos**. São Paulo : Blücher, 1979.

[COA92] COAD, Peter et YOURDON, Edward. **Análise baseada em objetos.** Rio de Janeiro : Campus, 1992.

[COL88] COLLINS,William J. **Programação estruturada com estudo de casos em PASCAL. /** Willian J. Collins; tradução Lisbete Madsen Barbosa. Rio de Janeiro : McGraw-Hill, 1988.

[DAM96] DAMASCENO JÚNIOR, Américo. **Aprendendo JAVA: programação na Internet.** São Paulo : Érica, 1996. 2ª ed.. 291p.

[FRA97] FRAIZER, Colin. **API Java : Manual de Referência. /** Colin Fraizer, Jill Bond; tradução e revisão técnica Álvaro Rodrigues Antunes. São Paulo : Makron Books, 1997. 371p.

[FUR73] FURTADO, Antônio Luz. **Teoria dos Grafos: algoritmos**. Rio de Janeiro : Livros Técnicos e Científicos, 1973.

[GUL96] GULBRANSEN, David. **Creating Web Applets With JAVA. /** David Gulbransen, Rawlings, Kenrick. Indianápolis : Sams.net Publishing, 1996. 307p.

[JOA93] JOÃO, Belmiro N. **Metodologias de desenvolvimento de sistemas.** São Paulo : Érica, 1993.

[LAL97] LALANI, Suleiman. **Java: biblioteca do programador. /** Suleiman “Sam” Lalani, Kris Jamsa; Tradução e revisão técnica Álvaro Rodrigues Antunes. São Paulo : Makron Books, 1997. 547p.

[NAU96] NAUGHTON, Patrick. **Dominando o JAVA. /** Patrick Naughton; tradução Kátia A. Roque; revisão técnica Álvaro Rodrigues Antunes. São Paulo : Makron Books, 1996. 425p.

[NEW97] NEWMAN, Alexander. **Usando Java: o guia de referência mais completo**. **/** Alexander Newman [et al]. Tradução Follow-UP traduções e Assessoria de Informática. Rio de Janeiro : Campus., 1997.

[RAB92] RABUSKE, Márcia Aguiar. **Introdução à Teoria dos Grafos.** Florianópolis : Editora da UFSC, 1992.

[RAB95] RABUSKE, Renato Antônio. **Inteligência Artificial.** Florianópolis : Editora da UFSC, 1995.

[SZW84] SZWARCFITER, Jayme Luiz. **Grafos e algoritmos computacionais**. Rio de Janeiro : Campus, 1984.