Aumentando a Eficiência da Solução de Problemas de Caminho Mínimo em SIG

Clodoveu Augusto Davis Jr.
Empresa de Informática e Informação do Município de Belo Horizonte S.A. - PRODABEL
Av. Presidente Carlos Luz, 1275
31230-000 - Belo Horizonte - MG
Tel.: (031)277-8391 - FAX: (031)462-3016
e-mail: `clodoveu@unix.horizontes.com.br`

RESUMO

O algoritmo que determina o menor caminho entre dois nós de uma rede é um clássico em ciência da computação. No entanto, apesar de ser bastante conhecido, sua implementação nos SIGs comerciais é bastante ineficiente, dando a falsa impressão que esta ineficiência é causada pela sua complexidade computacional. Este artigo apresenta o algoritmo do caminho mínimo de Dijkstra, e demonstra sua aplicação em problemas de roteamento urbano, tais como a escolha de rotas ótimas dentro da malha de circulação viária e no sistema de otimização de viagens utilizando transporte coletivo de Belo Horizonte. São comparados os tempos de execução do algoritmo usando os recursos de um SIG contra programas externos em C, otimizados para utilizar o máximo potencial dos microprocessadores atuais. Demonstra-se a viabilidade de desenvolver rotinas otimizadas específicas, interfaceando-as ao SIG, em detrimento do uso das rotinas internas usualmente encontradas.

ABSTRACT

The algorithm that determines the shortest path between two network nodes is a computer science classic. However, in spite of being well known, the implementation of such algorithm in commercial GIS software is rather inefficient, producing the false impression that this inefficiency is caused by the algorithm's computational complexity. This article presents Dijkstra's shortest path algorithm, and shows its application in urban routing problems, such as the generation of optimal routes within Belo Horizonte's traffic network and urban transportation optimal trip system. A comparison on the algorithm's execution time is made, between its implementation as a GIS feature and as external C programs that have been optimized to extract the most out of today's microprocessors. The viability of developing such specific, externally interfaced, optimized routines is demonstrated, as compared to the usual GIS internal routines.

INTRODUÇÃO

O problema de encontrar o caminho mais curto entre dois nós de um grafo ou uma rede é um dos clássicos da ciência da computação. Este problema consiste, genericamente, em encontrar o caminho de menor custo entre dois nós da rede, considerando a soma dos custos associados aos arcos percorridos[1].

O problema do caminho mínimo se adapta a diversas situações práticas. Em roteamento, por exemplo, pode-se modelar os nós do grafo como cruzamentos, os arcos como vias, e os custos associados aos arcos correspondem a tempo de trajeto ou distância percorrida, e a solução será o caminho mais curto ou o caminho mais rápido entre dois pontos. Em redes de computadores, os nós poderão representar equipamentos diversos, os arcos correspondem a trechos de cabeamento, e os custos poderão estar associados à taxa máxima de transmissão de dados. Neste caso, a solução será a rota de transmissão mais rápida. Outras possibilidades de aplicação incluem quaisquer problemas envolvendo redes ou grafos em que se tenha grandezas (distâncias, tempo, perdas, ganhos, despesas) que se acumulem linearmente ao longo do percurso da rede.

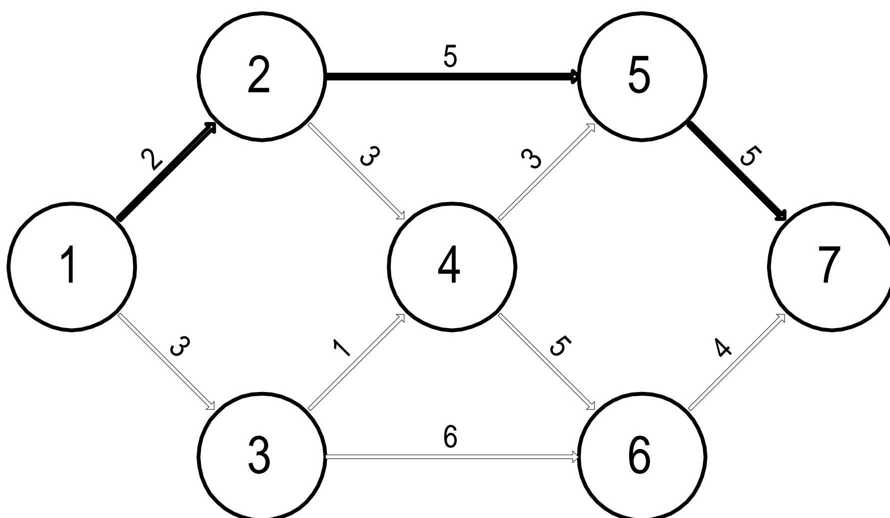

Figura 1 - Grafo e caminho mínimo (1-2-5-7, custo 12 unidades)

Existem algumas variantes deste problema, cada uma adequada a um conjunto de problemas diferente [1][2]:

- ***problema de único destino:*** consiste em determinar o menor caminho entre cada um dos nós do grafo e um nó de destino dado

---

[1] Embora este problema possa ser analisado considerando grafos não direcionados, este artigo abordará grafos direcionados (também chamados *dígrafos*), sempre considerando a possibilidade de associação de custos aos arcos.

- ***problema de única origem:*** determinar o menor caminho entre um nó dado e todos os demais nós do grafo
- ***problema de origem-destino:*** determinar o menor caminho entre dois nós dados
- ***problema de todos os pares:*** determinar o menor caminho entre cada par de nós presentes no grafo

Neste artigo será explorada a classe de problemas de origem-destino, pois é a que melhor se adapta aos problemas práticos que se busca resolver com o auxílio de um sistema de informações geográficas, em especial os problemas de roteamento em redes de transportes. Observe-se que existem outros tipos de problemas de interesse da área de roteamento que não se enquadram na classificação acima. O mais famoso deles é o problema do caixeiro-viajante, em que é dado um nó de origem e uma lista de nós de destino, e o objetivo é estabelecer a seqüência de percurso dos nós de destino de modo a minimizar o custo da viagem, visitando todos e retornando ao ponto de partida. Este problema é consideravelmente mais complexo que os problemas de caminho mínimo origem-destino, e é resolvido com a utilização de algoritmos de programação linear, baseados no método Simplex.

O ALGORITMO DE DIJKSTRA

Existem basicamente dois algoritmos para resolver o problema de caminho mínimo em redes com origem e destino dados. O primeiro, e mais famoso deles, é o algoritmo de Dijkstra, proposto em 1959 e aperfeiçoado desde então principalmente pelo uso de estruturas de dados mais eficientes. Este algoritmo, que será apresentado em mais detalhes a seguir, apenas funciona se os custos associados aos arcos forem não negativos. O outro algoritmo, proposto separadamente por Bellman (1958) e Ford (1962) admite o caso mais geral de custos negativos. Como, em geral, as grandezas físicas associadas aos arcos são naturalmente não negativas, o algoritmo de Dijkstra acaba por se constituir no método de solução de problemas de caminho mínimo mais empregado na prática.

O algoritmo considera um grafo G, composto por um conjunto de nós, denominado N, e um conjunto de arcos denominado A. Além disto, são conhecidos dois nós pertencentes a N, denominados o e d, que são respectivamente a origem e o destino. Os nós são divididos em três grupos: os já visitados (conjunto V), os candidatos ou de fronteira (conjunto F) e os nunca visitados ou "desconhecidos" (conjunto D). O conjunto V é inicializado para conter apenas o nó o. Os vizinhos imediatos de o são colocados no conjunto F, sendo registrados os custos para atingí-los a partir de o, e os demais inicialmente pertencem ao conjunto D. A cada passo do algoritmo, os nós de F são verificados para determinar qual seria a melhor opção para expandir a pesquisa. Será escolhido e transferido para V aquele nó cujo custo acumulado seja o menor dentre os candidatos, e seus vizinhos serão então transferidos do conjunto D para o conjunto F. A pesquisa para quando o nó d for

alcançado ou quando não houver mais nós a percorrer (neste caso, não existe caminho viável entre *o* e *d*).

A figura 2 apresenta o algoritmo codificado esquematicamente [2]. Observe-se que, da forma apresentada, admite-se que todos os dados necessários estejam organizados em memória segundo estruturas de dados específicas. Os nós estão armazenados no vetor `nodos`, que tem diversos campos: `sit` (situação do nó, ou a qual conjunto o nó pertence), `acum` (custo acumulado até o nó) e `pai` (nó anterior no caminho mínimo). Os arcos estão organizados em uma lista de adjacência, cujo ponto inicial está armazenado no campo `adj` do vetor `nodos`. O conjunto *F* é também representado por uma lista, denominada `lista_fronteira`.

A codificação das estruturas de dados é, portanto, a seguinte:

```
struct NODO
{
   INT id;      /* identificacao do nó */
   INT x;       /* coordenada geográfica x do nó */
   INT y;       /* coordenada geográfica y do nó */
   INT acum;    /* custo acumulado */
   INT pai;     /* nó anterior (resultado) */
   unsigned char sit;  /* situacao do nó */
   struct ADJ *adj;    /* lista de adjacência */
};
struct NODO nodos[MAXNODOS]; /* vetor de nós */

struct ADJ
{
   INT nodo;         /* no adjacente */
   INT idarc;        /* identificador do arco */
   INT custo;        /* custo associado ao arco */
   struct ADJ *link; /* continuação da lista */
};
```

O critério de seleção do próximo nó a visitar é crucial para o funcionamento do algoritmo. A escolha é feita considerando o nó cuja “aquisição” minimize, naquele passo, a expansão do custo acumulado até aquele ponto. Este critério é típico dos algoritmos ditos *gulosos* (*greedy*), e embora seja difícil acreditar à primeira vista, o método realmente conduz à escolha do caminho de menor custo, conforme demonstrado em [1]. A expansão da pesquisa pode ser melhor visualizada em uma rede com características geográficas. O padrão de expansão se assemelha a um “borrão de tinta”, pois tende a se espalhar de forma aproximadamente circular em torno do ponto de origem (figuras 3 e 4).

```
CaminhoMínimo(nodos, n, o, d)
Início
    para i = 0 até n faça
        nodos[i].sit = D;
    nodos[o].acum = 0;
    nodos[o].sit = V;
    x = o;
    semsolução = false;
    enquanto x != d e não semsolução faça
        ptr = nodos[x].adj;
        enquanto ptr != nil faça
            y = ptr->nodo;
            se nodos[y].sit = F e
                (nodos[x].acum + ptr->custo) <
                    nodes[y].acum)
            então
                nodos[y].pai = x;
                nodos[y].acum = nodos[x].acum +ptr->custo;
            fim se;
            se nodos[y].sit = D então
                nodos[y].sit = F;
                nodos[y].pai = x;
                inserir y no início da lista_fronteira;
                nodos[y].acum = nodos[x].acum +ptr->custo;
            fim se;
            ptr -> ptr->link;
        fim enquanto;
        se lista_fronteira = nil
        então semsolução = true;
        senão
            percorrer a lista_fronteira e encontrar
                o nó com menor nodos[i].acum;
            x = i tal que
                nodos[i].acum seja mínimo em
                lista_fronteira;
            remover x da lista_fronteira;
            nodos[x].sit = V;
        fim se;
    fim enquanto;
    apresentar saída: d até o em ordem reversa;
fim CaminhoMínimo.
```

Figura 2 - Algoritmo de Dijkstra

Analisando o algoritmo, verifica-se que sua ordem de complexidade é $O(n^2)$, onde n é o número de nós. Ou seja, o tempo de execução cresce, na pior das hipóteses, com o quadrado do número de nós. Isto corresponde à definição original do algoritmo por Dijkstra. Trabalhos posteriores demonstraram que a complexidade pode ser reduzida para $O(m \log_2 n)$, onde m é o número de arcos, simplesmente substituindo a lista que recebe os nós do conjunto F por uma fila de prioridades [3].

IMPLEMENTAÇÃO EM UM SIG

Conforme já dito, o algoritmo segundo apresentado até aqui utiliza dados que já estão previamente organizados em estruturas de dados na memória. Na implementação de pesquisas de caminho mínimo em SIG, existe a complexidade adicional de selecionar e extrair os dados necessários não da memória, mas de um banco de dados. Certamente o uso da memória principal estará reservado para atividades mais críticas para a performance do sistema como um todo, funções que serão usadas mais frequentemente, tais como rotinas de *display*, indexação espacial e interface com o usuário. Além disto, para montar em memória as estruturas de dados mais eficientes para resolver o problema do caminho mínimo, seria necessário selecionar todos os nós e arcos do grafo no banco de dados, recuperando dados alfanuméricos associados, e organizá-los nas estruturas, antes que se pudesse iniciar o processamento. Caso o caminho a pesquisar esteja restrito a uma pequena região, ainda assim toda a estrutura precisa ser recuperada, pois não se sabe *a priori* qual será o caminho de expansão da pesquisa.

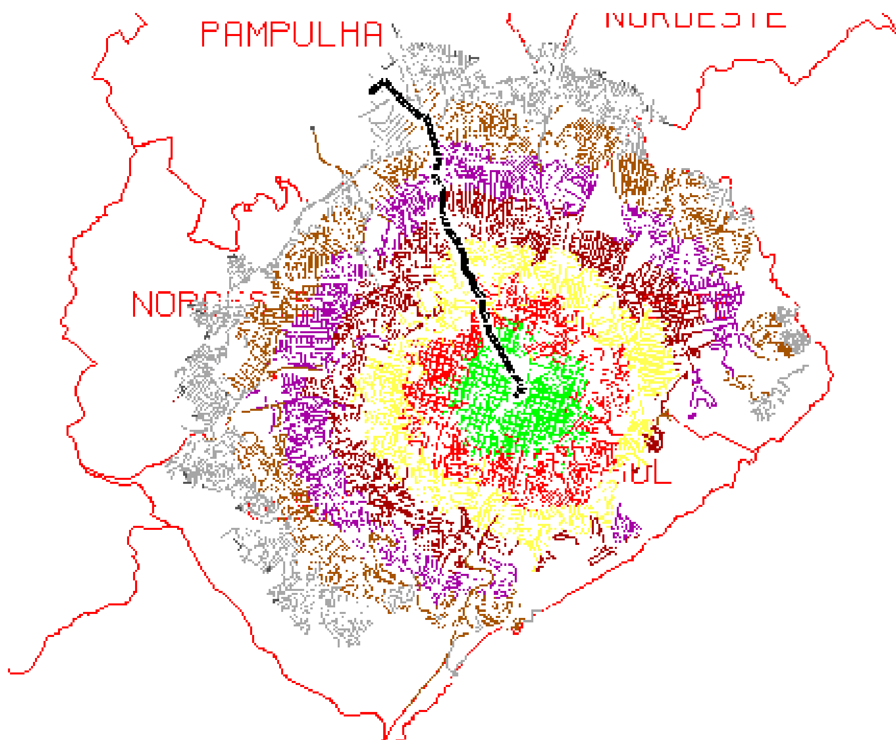

Figura 3 - Padrão de expansão da pesquisa no algoritmo de Dijkstra - Caso 1
(cada variação de cor corresponde a 5.000 objetos visitados)

Portanto, a opção natural para a implementação deste recurso seria realmente fazer a recuperação dos dados à medida em que sejam demandados pelo algoritmo, na esperança de recuperar o mínimo possível de informações, e ao mesmo tempo restringir o uso da memória principal ao mínimo indispensável. O algoritmo oferece uma oportunidade ideal para isto, exatamente no ponto em que o nó é transferido para o conjunto V e sua vizinhança é colocada no conjunto F. Infelizmente, isto implica em uma performance bem mais baixa que a que poderia ser alcançada com o uso do algoritmo otimizado, com os dados em memória.

A opção pela recuperação do grafo a partir do banco de dados também tem mais duas razões de ordem prática. Uma delas é a possibilidade de se ter grafos maiores que a capacidade de memória do equipamento, o que hoje em dia só ocorreria em bases muito grandes. Outra é a necessidade de suportar um ambiente multiusuário, em que um operador poderia estar realizando manutenção em elementos do grafo enquanto outro usuário estaria tentando resolver um problema de caminho mínimo. Neste caso, a existência de uma cópia do grafo em memória poderia resultar em uma solução errônea, ou inconsistente com o estado do banco de dados.

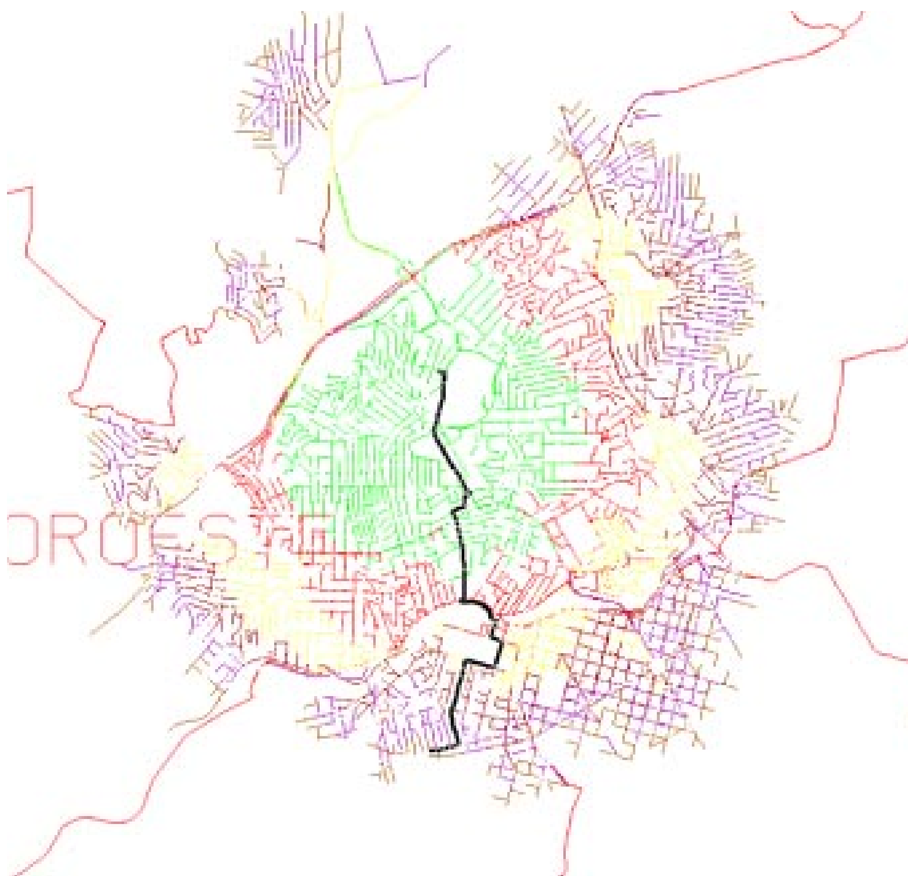

Figura 4 - Padrão de expansão da pesquisa no algoritmo de Dijkstra - Caso 2
(cada variação de cor corresponde a 2.000 objetos visitados)

Desta forma, a decisão do projetista do SIG é praticamente inevitável: sacrificar a performance da solução de problemas de caminho mínimo (que são executados raramente) em detrimento de (1) operações realizadas com mais freqüência e que necessitam da memória como um recurso crítico; (2) possibilitar a solução de problemas virtualmente sem limitações quanto ao tamanho do grafo; e (3) garantir a integridade da solução em ambientes de processamento concorrente.

AUMENTANDO A EFICIÊNCIA

Existem situações práticas, em determinadas aplicações, em que os tempos de resposta que se pode conseguir na solução de problemas de caminho mínimo em SIG simplesmente se tornam inaceitáveis. Será proposta a seguir uma alternativa de implementação que busca melhorar os tempos de solução de problemas, mantendo a compatibilidade com o SIG.

A alternativa consiste em implementar o algoritmo de Dijkstra em C, externamente ao SIG, preparando-o de modo que possa utilizar dados extraídos da base geográfica e produzir resultados que possam ser realimentados ao SIG. No SIG, é necessário desenvolver uma rotina que possa recuperar todas as informações do grafo a partir do banco de dados, produzindo arquivos que serão lidos pelo programa em C externo. Desta forma, será possível extrair toda a velocidade de processamento possível para o algoritmo de Dijkstra, usando todos os recursos de estruturação de dados e otimização de código disponíveis. Além disto, o SIG deve ser preparado para receber os resultados e apresentá-los ao usuário.

A principal limitação que se vai impor é quanto à freqüência de atualização do grafo na base geográfica. Será pouco prático fazer a exportação do grafo para arquivos externos constantemente, pois esta operação pode ser a mais demorada. Assim, o método proposto aqui se adaptará melhor a situações em que as modificações no grafo são esporádicas, não sendo crítico para a aplicação manter o grafo rigorosamente atualizado.

RESULTADOS PRÁTICOS

São apresentados a seguir os resultados da implementação do algoritmo do caminho mínimo para resolver problemas de otimização de rotas dentro da rede de circulação viária de Belo Horizonte. Esta rede, que conta atualmente com mais de 35.500 nós e 89.000 arcos, está disponível na base de dados do software APIC, que é o SIG adotado pela Prefeitura de Belo Horizonte.

Este SIG tem, dentre as diversas funções implementadas como comandos de sua linguagem de programação, um comando capaz de determinar o caminho mínimo dentro de uma rede. No caso da rede de trânsito de Belo Horizonte, no entanto, a performance deste comando no ambiente SIG é sofrível, com tempo de resposta que inviabiliza boa parte das aplicações práticas que a rede de trânsito poderia ter. A implementação produzida para este trabalho provou-se capaz de prover uma alternativa viável para a resolução de problemas de caminho mínimo em cooperação com o ambiente SIG, dele extraindo dados e gerando resultados que podem facilmente ser recebidos pelo SIG, para pós-processamento ou apresentação.

Os dados da rede, extraídos do SIG, são codificados em um arquivo alfanumérico usando uma estrutura de lista de adjacências, em que a primeira parte ocorre uma vez para cada nó, e a parte entre colchetes se repete uma vez para cada arco que sai do nó, da seguinte forma:

```
ident.nó, coord. X do nó, coord. Y do nó
  [,ident. nó sucessor, ident. arco, custo]
```

Onde:

`ident.nó`	identificador único do objeto correspondente ao nó na base de dados geográfica
`coord. X do nó`	coordenada UTM X (Leste) do nó
`coord. Y do nó`	coordenada UTM Y (Norte) do nó
`ident. nó sucessor`	identificador único do nó sucessor
`ident. Arco`	identificador único do arco que conecta os nó em questão ao seu sucessor
`custo`	custo do arco (nos testes, corresponde ao comprimento em metros do arco, arredondado para o inteiro mais próximo)

Os identificadores únicos citados acima constituem uma chave de acesso ideal para os objetos arco e nó na base de dados geográfica do APIC. Como os relacionamentos entre estes objetos na base geográfica são construídos de forma a configurar a rede, torna-se fácil recuperar a lista de adjacência de cada nó, e incorporá-la ao arquivo. No entanto, cada um destes identificadores na lista de adjacência precisa ser substituído, para que o algoritmo funcione, por referências ao número de ordem do nó no vetor que os armazena na memória. Para que isto seja conseguido, o programa implementa uma rotina de tradução, baseada numa árvore binária de pesquisa, que estabelece a correspondência entre o identificador único de cada nó e o número de ordem no vetor, que varia de 0 a $(n - 1)$.

Em seguida, o programa precisa receber como dados de entrada a identificação dos nós que serão a origem e o destino do caminho mínimo. Esta identificação é obtida através de uma interação com o usuário no ambiente geográfico, em que todos os recursos para localização estão disponíveis: endereços, centerlines, cruzamentos, bairros, pontos de referência. Os identificadores obtidos são passados para o programa pela própria linha de comando, usando um recurso do APIC que permite a execução, de dentro de seu ambiente, de comandos do sistema operacional. Existe ainda a possibilidade de enviar estes dados e coletar os resultados via datagramas UDP/IP, usando os recursos do APIC para comunicação interprocessos.

Em todos os testes relatados a seguir, foi utilizada uma estação de trabalho IBM RS/6000, modelo 320H, lançada em 1991, com 64 Mbytes de memória RAM, sendo que sua CPU é capaz de desempenho da ordem de 60 SPECint92 [4]. Este indicador de performance é semelhante ao de um microprocessador Pentium de 100 MHz atual, embora a estação de trabalho tenda a levar vantagem em relação a microcomputadores comuns na velocidade de acesso a disco.

A Tabela 1 apresenta os resultados comparativos entre a execução de dois casos diferentes, usando a rotina de caminho mínimo interna ao APIC e usando o programa externo.

Tabela 1
Comparação de Tempos de Resolução de Problemas de Caminho Mínimo

	APIC	**Algoritmo de Dijkstra Clássico**	**Fator de Aceleração (aproximado)**
Tempo Caso 1 (*)	1580 s	5,8 s	270 vezes
Tempo Caso 2 (**)	421 s	1,35 s	310 vezes

(*) Média de 20 execuções
(**) Média de 40 execuções

O Caso 1 corresponde a uma rota de aproximadamente 10 km (figuras 3 e 5), e o Caso 2 corresponde a uma rota de aproximadamente 5 km (figuras 4 e 6), ao longo da qual existe um obstáculo físico que precisa ser contornado. Como se pode perceber, o tempo de execução do programa externo é expressivamente mais baixo que o do comando do SIG, conforme esperado. O desempenho do APIC é prejudicado, além dos fatores ligados ao acesso a banco de dados, pela necessidade de acesso cliente-servidor através de uma rede local. O tempo de execução do algoritmo de Dijkstra também não considera o tempo necessário para ler o arquivo de entrada e formar as estruturas de dados na memória, que é de aproximadamente 21 segundos. Optou-se por não incluir este tempo nas medições porque o caso real de execução fará esta carga uma única vez para a resolução de diversos problemas usando o mesmo grafo.

HEURÍSTICA DE HART-NILLSON-RAPHAEL

Apesar dos expressivos resultados obtidos no teste relatado acima, existem ainda oportunidades para aperfeiçoamentos. O algoritmo de Dijkstra foi desenvolvido para resolver problemas em redes genéricas. No caso da rede de circulação de trânsito, seria interessante utilizar a informação de que dispomos a respeito da distribuição geográfica da rede.

No algoritmo padrão de Dijkstra, escolhe-se como próximo nó aquele para o qual o custo acumulado será o menor. A heurística de Hart-Nillson-Raphael propõe que esta escolha seja feita de modo que o nó escolhido seja aquele para o qual a soma do custo acumulado e da distância euclidiana do nó candidato até o nó destino seja a menor. Esta estratégia consegue reduzir significativamente o número de nós visitados, modificando o padrão de expansão da pesquisa do já citado “borrão de tinta” para um padrão direcional, que caminha mais facilmente na direção aproximada do nó de destino (figura 5, correspondente ao Caso 1, e figura 6, correspondente ao Caso 2).

A Tabela 2 apresenta uma comparação da execução dos dois casos anteriormente apresentados, usando o algoritmo de Dijkstra clássico e o algoritmo modificado com a inclusão da heurística de Hart-Nillson-Raphael. O ganho adicional

em tempo de processamento é decorrência do menor número de nós visitados. Em contrapartida, é necessário calcular uma distância geográfica a cada passo do algoritmo, o que acrescenta custo computacional pela inclusão de operações de ponto flutuante.

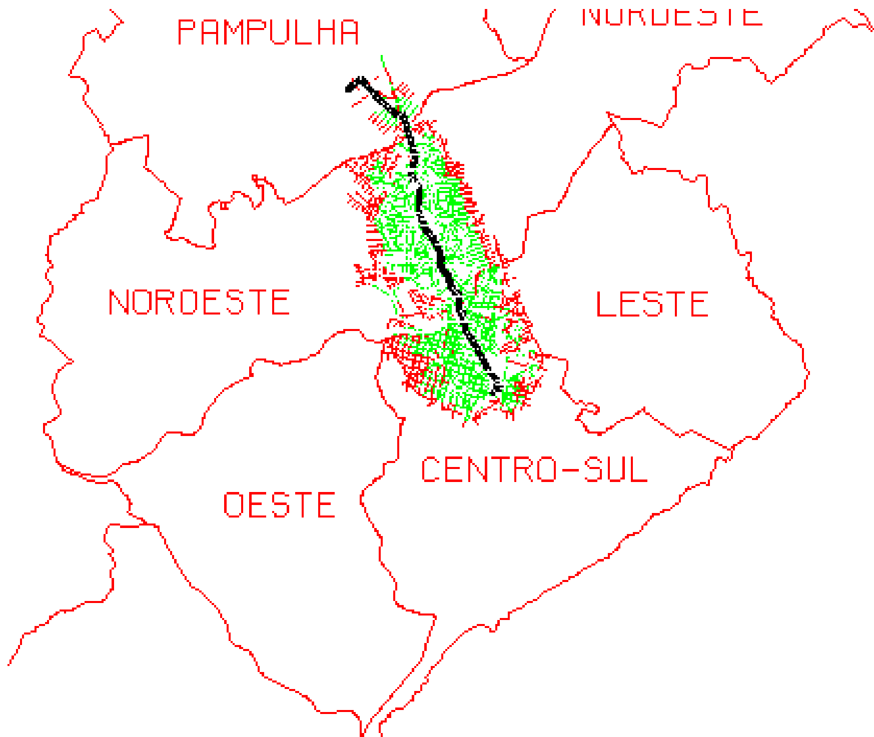

Figura 5 - Padrão de expansão da pesquisa no algoritmo de Dijkstra usando a heurística de Hart-Nillson-Raphael - Caso 1
(cada variação de cor corresponde a 5.000 objetos visitados)

É importante ressaltar que a solução encontrada pelas três alternativas (APIC, algoritmo clássico e algoritmo modificado) é rigorosamente a mesma.

Além dos casos citados, foram realizados ainda diversos testes de traçado de rotas ótimas utilizando a rede de trânsito de Belo Horizonte, para uma avaliação qualitativa dos resultados. O objetivo era obter caminhos mínimos em trajetos habituais de pessoas com bom conhecimento da cidade, e comparar os resultados com os caminhos preferidos por estas pessoas. Na maior parte dos casos, o caminho mínimo obtido se aproximou bastante do caminho preferido pelas pessoas, sendo que as diferenças se deveram principalmente à preferência pessoal pelo tráfego em avenidas, em detrimento a atalhos por ruas de menor movimento (e pior pavimentação). Outras diferenças entre os caminhos preferidos e os caminhos mais

curtos podem ser atribuídas à existência de semáforos e regiões de tráfego intenso, levando a um tempo de viagem mais longo, apesar do caminho mais curto.

Tabela 2
Comparação de Tempos de Resolução de Problemas de Caminho Mínimo

	APIC	**Algoritmo de Dijkstra Clássico / Aceleração**	**Algoritmo com Heurística / Aceleração**
Tempo Caso 1 (*)	1580 s	5,8 s / 270 vezes	1,3 s / 1200 vezes
Objetos Visitados Caso 1	N.D. (***)	35.088	7.234 (20,6 %)
Tempo Caso 2 (**)	421 s	1,35 s / 310 vezes	0,2 s / 2100 vezes
Objetos Visitados Caso 2	N.D. (***)	11.062	2.540 (23,0 %)

(*) Média de 20 execuções
(**) Média de 40 execuções
(**) Apesar de não ser possível verificar este número no APIC, supõe-se que seja idêntico ao do algoritmo clássico

CONCLUSÕES

A comparação do tempo de processamento entre as duas alternativas favorece claramente o uso da heurística. No caso do teste, reduziu-se em cinco vezes o número de objetos acessados, indicando que, se esta técnica estivesse sendo utilizada no algoritmo implementado no SIG, o tempo de processamento naquele ambiente seria bem menor.

Mesmo sem a heurística, a aceleração obtida com relação ao processamento no SIG, na mesma máquina, varia entre 200 e 400 vezes. Resultados comuns, nas estações de trabalho utilizadas, mostram a obtenção de soluções em 3-20 segundos, contra 5-120 minutos utilizando o SIG.

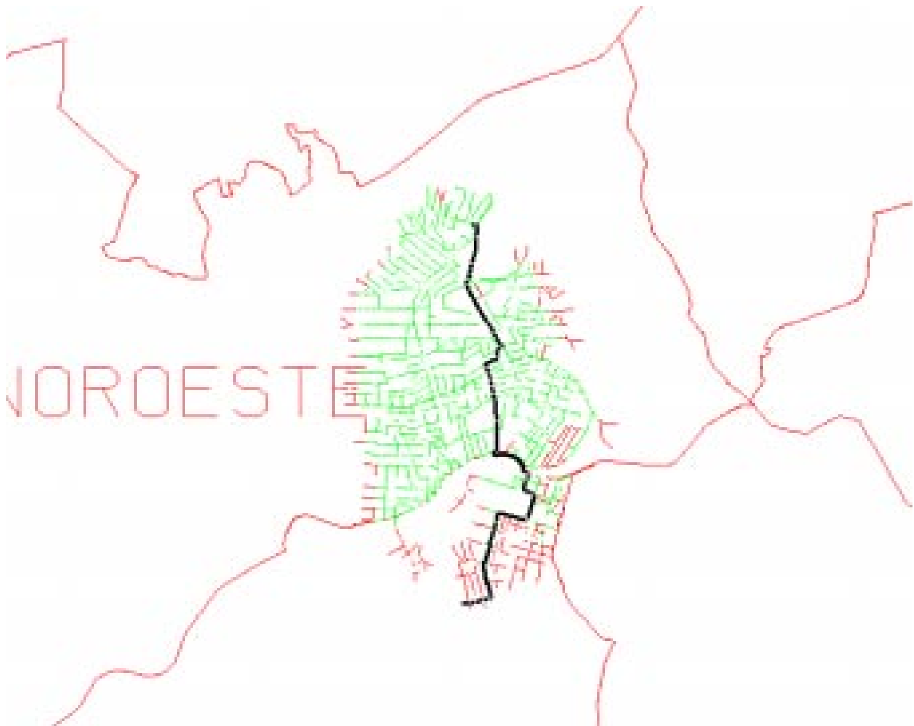

Figura 6 - Padrão de expansão da pesquisa no algoritmo de Dijkstra usando a heurística de Hart-Nillson-Raphael - Caso 2
(cada variação de cor corresponde a 2.000 objetos visitados)

Naturalmente, esta sensível redução de tempo se deve a muitos fatores, tais como (1) a disponibilidade de características mais genéricas no algoritmo do SIG, (2) o processamento inteiramente em memória no caso deste programa, (3) a necessidade de acesso a elementos de rede por meio de seleções em ambiente de banco de dados, no caso do SIG, entre outros. No entanto, os tempos obtidos são expressivamente baixos utilizando equipamento de baixo custo, indicando que diversas aplicações importantes poderão ser construídas a partir desta implementação. Entre elas, destacamos:

- despacho de viaturas de emergência (ambulâncias, corpo de bombeiros), com orientação via rádio;
- fornecimento de informações sobre rotas por telefone, para cidadãos ou turistas;
- otimização de rotas usando transporte coletivo (ônibus e metrô) [5].

Os resultados obtidos mostram, também, como seria grande o benefício de se ter a heurística de Hart-Nillson-Raphael implementada no algoritmo do SIG, pelo menos como opção. Para os casos em que a aplicação pudesse se beneficiar da heurística, a redução do número de objetos visitados poderia causar reduções

expressivas nos tempos de processamento: nos casos apresentados, esta redução poderia ser de até 80%.

Com relação à aplicação prática de escolha do melhor caminho (em lugar do caminho mais curto) na malha viária urbana, seria necessário levar em conta penalidades relativas a semáforos, pavimentos de pior qualidade, conversões e volume de tráfego. Neste caso, o mais interessante seria transformar os custos, que nos testes foram expressos em metros, para tempo. O SIG pode fazer isto com facilidade, calculando os tempos de percurso de cada trecho de via a partir do comprimento e de outras informações da malha de logradouros. Seria necessário:

- adotar uma velocidade média em cada trecho. Para simplificar, cada tipo de logradouro poderia ter uma velocidade média: mais alta nas avenidas, menor nas ruas, ainda menor nas praças, e assim por diante;
- adotar velocidades bem baixas em trechos de conversão, como forma de penalizar a mudança brusca de direção;
- adotar um tempo médio por semáforo, e somá-lo ao tempo de percurso do trecho precedente.

O mais difícil seria incorporar penalidades devido ao volume médio de tráfego, pois este dado teria que ser obtido em campo, e deveria ser avaliado por faixa de horário. Mesmo assim, apenas as transformações acima seriam suficientes para fazer com que as rotas sejam mais realistas, viabilizando as aplicações citadas.

REFERÊNCIAS

[1] Cormen, T.H., Leiserson, C.E. and Rivest, R.L., *Introduction to Algorithms*, McGraw-Hill and MIT Press, Cambridge, Mass., 1990.

[2] Baase, Sara *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, Second Edition, 1988.

[3] Tarjan, Robert E. *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics (SIAM), 1983.

[4] Hennessy, John L. and Patterson, David A., *Computer Architecture: A Quantitative Approach*, Second Edition, Morgan Kaufmann, 1996.

[5] Zuppo, Carlos A., Davis Jr., Clodoveu A. e Meirelles, Alexandre A. C. *Geoprocessamento no Sistema de Transporte e Trânsito de Belo Horizonte*, Anais do GIS Brasil'96 (376-387), 1996.