Algoritimia e Desempenho em Redes de Computadores

Inter-AS Routing and Path Inflation

Eduardo Rodrigues nº 86984
Tomás Malcata nº 87130

Prof. João Sobrinho

Turno: Quarta às 12h30
Grupo 7

1 Introdução

Neste segundo projecto, foi estudado o funcionamento do tráfego de dados através da Internet, mais concretamente, através das redes constituintes da Internet, denominadas de *Autonomous Systems (AS)*. Para além do mais, foram implementados diversos algoritmos de modo a simular o funcionamento do protocolo de roteamento entre sistemas autónomos, o *Border Geteway Protocol (BGP)*. Protocolo este que para cada *AS* elege uma rota para o destino, sempre tendo em conta as relações comerciais entre cada duas *ASes* vizinhas, e de modo a que cada *ISP (Internet Service Provider)* faça lucro com o tráfego de dados. Consequentemente, ao longo do projecto teve que se ter em conta as políticas de roteamento entre *ASes*, o que se traduz num efeito característico do BGP, o *path inflation*. Estas políticas podem ser resumidas na tabela 1.

Tabela 1: Regras de exportação de rotas.

		Recebida de		
		Client	Peer	Provider
Exporta a	Client	Sim	Sim	Sim
	Peer	Sim	Não	Não
	Provider	Sim	Não	Não

O programa foi desenvolvido em linguagem C, tendo como entrada ./main file, onde file é o nome do ficheiro com a rede a analisar.

2 Estruturas Utilizadas

Visto que a Internet e as suas *ASes* constituintes foram fornecidas sob a forma de um ficheiro de texto, onde estavam explicitadas as relações comerciais entre cada par de *ASes* vizinhas, utilizou-se uma lista de adjacência de maneira a implementar os algoritmos pedidos. Foi feita a assumpção de que não haveria *ASes* com valor superior a (2^{16}), pelo que esta lista de adjacência é composta por um vector de dimensão 65536 onde o índice corresponde ao número da *AS*. Deste modo a procura de uma *AS* específica tem complexidade O(1)

Dito isto, é importante evidenciar que a estrutura de cada entrada do vector *graph[AS]* é composta por quatro inteiros e três apontadores. Mais concretamente, um inteiro que identifica a AS em questão, um segundo inteiro (*type*) que pode tomar três valores: 1 caso seja uma *AS* sem fornecedores, ou seja, um *Tier-1* (T1), 3 para *ASes* sem clientes (stub) e, por fim, 2 para todas as outras *ASes*, o terceiro e quarto inteiros (*visit* e *open*) são auxiliares para a realização dos algoritmos. A última parte da estrutura *Graph* é composta pelos apontadores e são as listas de adjacência propriamente ditas , cada uma formada por uma lista ligada com os nós da estrutura do tipo *Succs* (*succedent*) onde é guardado o valor da *AS* pretendido e um apontador para o próximo nó. Cada uma destas listas guarda informação sobre as ligações *peer, client e provider* dessa AS. O vector principal foi alocado previamente sendo as *ASes* e as listas de adjacência inicializada à medida que se lê o ficheiro de entrada.

3 Verificações iniciais

As redes obtidas no ficheiro de entrada são à partida comercialmente conexas e não têm ciclos iniciais. Ou seja, é sempre possível a partir de uma determinada *AS*, chegar a outra *AS* através de rotas comerciais válidas, e nunca é possível partir de uma AS para chegar à mesma, apenas através de rotas do tipo cliente.

A conectividade é necessária para garantir a possibilidade de comunicação entre todas as *AS* e a ausência de ciclos é necessária para que o protocolo *BGP* atinja um estado de estabilidade (sem alterações de rotas). Caso estas propriedades não se verifiquem, seria necessário colocar as ligações em falta (para o caso da conectividade) e retirar as ligações que criam ciclos. Porém, estas operações não foram implementadas, tendo sido apenas verificado se estas propriedades eram respeitadas.

3.1 Ciclos comerciais

De modo a verificar se existem ciclos, foi efectuada uma *BFS* apenas através das ligações do tipo *client*. Para tal, é necessário utilizar os parâmetros *visit* e *open* da estrutura *graph*.

Foi utilizado o algoritmo . Este algoritmo percorre um determinado caminho colocando o parâmetro *open* a 1 sempre que passa numa *AS*. Caso encontre uma *AS* já aberta, o algoritmo encontra um ciclo, logo a função retornará 1. Para além do mais, sempre que se começa o algoritmo numa determinada *AS* esta é marcada como visitada.

Por fim, este algoritmo é repetido em todas as *ASes* que ainda não foram visitadas. Ou seja, este é realizado na primeira *AS* e posteriormente será realizado na segunda, caso esta não tenha sido já marcada como visitada. O algoritmo efectuado percorre uma vez cada vértice (se percorrer a segunda será um ciclo e parará), pelo que tem complexidade O(V), onde V é o número de *ASes*.

Algorithm 1 CheckCycle

```
1:  procedure CheckCycle(graph,AS)
2:      p ← 0
3:      graph[AS].visit ← 1
4:      graph[AS].open ← 1
5:      for aux in all AS clients do
6:          i ← Aux− > AS
7:          if graph[i].open then return 1
8:          if graph[i].visit= 0 then
9:              p ← CheckCycles(graph, i)
10:     graph[AS].open ← 0
11:     return p
```

3.2 Conectividade

Para verificar esta segunda propriedade é verificada se todas as *ASes* to tipo *T1* têm ligações do tipo *peer* com todas as outras *ASes*. Tal é verificado de forma simples, uma vez que aquando a inserção de novas ligações, as *ASes* são actualizadas consoante o seu tipo (se são *T1*, *stubs* ou um outro *tier* intermédio). Esta actualização faz-se tendo por base que as *AS* do tipo *T1* só têm ligações do tipo *client* e os *stubs* só têm

ligações do tipo *provider*. Deste modo as *ASes* são inicializadas com o tipo *stub* ou *T1* (consoante a primeira ligação seja do tipo *provider* ou *client*). Por fim, se a um *stub* se colocar uma relação do tipo *client*, passa a ser do tipo *tier* intermédio. O oposto será efectuado para os *Tier 1*.

Em suma, o programa tendo estas informações, conta o numero de *T1* existentes e verifica para cada um deles se o numero de *peers* que são *T1* corresponde ao numero total de *peers* menos 1. No pior caso este algoritmo tem complexidade de O(V$\hat{2}$), visto que numa rede apenas com *T1* (que comercialmente não tem significado pois não há transferência de dinheiro) percorre cada *AS* duas vezes.

4 Algoritmos principais

De modo a encontrar os melhores tipos de rotas e os caminhos mais curtos entre todas as *ASes*, baseamos-nos no algoritmo de Dijkstra. O primeiro algoritmo encontra o melhor tipo de rota comercial com a menor distância possível de todas as *ASes* para uma determinada *AS*, enquanto que o segundo encontra a menor distância (sem ter que respeitar o protocolo BGP) de todas as *ASes* para uma determinada *AS*. De notar que o propósito destes algoritmos é realizar estatísticas e não guardar a informação, logo, estes não alocam memória em cada nó do grafo para guardar informação dos tipos de rotas e tamanhos, mas actualizam as estatísticas em tempo real.

4.1 Estatísticas

Para a realização das estatisticas foram criadas 5 variáveis globais no stats.c, dois vectores de inteiros (*length_commercial* e *length_shortest*) que guardam o número de rotas com a dimensão dos seus índices (por exemplo, para o índice 0 estará o número de rotas com dimensão 0) para as rotas comerciais e não comerciais respectivamente. Foram também criadas 3 variáveis *providers*, *providers* e *providers* que guardam o número de ligações do respectivo tipo.

Deste modo, no primeiro algoritmo o vector *lenght_commercial* e as váriaveis são actualizadas através da função *update_com mercial_stats(size,type)* (que incrementa o tipo de rota e tamanho correspondente) e no segundo algoritmo o vector *lenght_shortest* é actualizado através da função *update_vector_shortest(size)*.

Por fim, as percentagens do tipo de rotas são efectuadas através das variáveis (*clients*, *peers* e *peers*). E as funções cumulativas a partir dos vectores.

4.2 Dimensão das rotas comerciais e tipos de rotas

Para se obter a dimensão das rotas comerciais e o tipo de rotas foi desenvolvido o algoritmo 2. Este utiliza uma estrutura auxiliar, um acervo de prioridades mínimas indexado. Para tal, considera-se mais prioritária a rota mais conveniente, ou seja, a que tiver o melhor tipo de rota sendo este representado pelos valores 1, 2 e 3 para rotas do tipo *client*, *client* e *provider*, respectivamente. Caso o tipo de rota seja o mesmo, é mais prioritário a rota que tiver menor tamanho. Deste modo, existem duas funções que realizam operações no acervo: *insert(AS, size)* e *RemovePrior(&AS, &type, &size)*. A primeira função irá inserir no acervo os valores, verificando se essa *AS* já existe no acervo e se os valores a inserir são os mais prioritários. Caso seja mais prioritário, procede-se à substituição dos novos valores no acervo (não existindo entradas repetidas no acervo), de notar que o facto de haver um vector extra que guarda as posições das *ASes* no acervo faz com que esta comparação de prioridades tenha complexidade O(1). Se a *AS* não existir no acervo esta será inserida. A segunda função apaga o nó mais prioritário e passa os seus valores para as variáveis *AS*, *type* e *size* por referência, esta retornará 0 se o acervo estiver vazio.

As restantes operações são a execução do protocolo BGP, ou seja, as rotas do tipo *client* (1) são anunciadas a todos os tipos de ligações enquanto que as rotas do tipo *peer* (2) e *provider* (3) apenas são anunciadas a clientes.

Sempre que um nó é visitado (visto que é sempre analisada a *AS* mais prioritária), o seu tipo de rota e tamanho encontrado será o mínimo possível. Posteriormente as variáveis das estatísticas são actualizadas. De notar que como a Internet é comercialmente conexa, é sempre possível encontrar um caminho de uma AS para outra.

É possível concluir que este algoritmo (de todas as *ASes* para uma *AS*) tem complexidade O((V + E) log V), onde V é o número de *ASes* do grafo e E o número de ligações. Tal verifica-se pois a inserção e remoção de elementos num acervo cheio, no pior caso tem custo de log(V) e que a remoção é executada uma vez por cada vértice (tendo O(V log (V))). Para além do mais, no pior caso, são inseridas todas as arestas somando o termo O(E log V). De notar que não são inseridos nós já visitados nem nós que já estejam no *heap* (podendo estes ser substituídos se o valor a inserir for melhor), fazendo com que o termo O(E log V) tenha uma diminuição de peso. Para além do mais existem arestas que não respeitam o protocolo BGP (caso a ligação não seja do tipo *client* (1) não é exportada para *peers* e *providers*), pelo que nem todas as são analisadas pelo algoritmo.

Algorithm 2 TypeOfRoutesWSize

```
1:  procedure TypeOfRoutesWSize(graph,AS)
2:      Insert(AS, 1, 0)
3:      while RemovePrior(&AS_t, &type, &size) do
4:          if graph[AS_t].visit != AS then
5:              graph[AS_t].visit ← AS
6:              update_commercial_stats(size,type)
7:              if type = 1 then
8:                  for aux in all AS_t providers do
9:                      if graph[aux->AS].visit = AS then
10:                         continue
11:                     Insert(aux− > AS, 1, size + 1)
12:                 for aux in all AS_t peers do
13:                     if graph[aux->AS].visit = AS then
14:                         continue
15:                     Insert(aux− > AS, 2, size + 1)
16:             for aux in all AS_t clients do
17:                 if graph[aux->AS].visit = AS then
18:                     continue
19:                 Insert(aux− > AS, 3, size + 1)
```

4.3 Rotas não comerciais

Para a segunda funcionalidade foi desenvolvido um algoritmo de Dijkstra para grafos sem pesos, sendo este idêntico a uma BFS (mudando o facto de inserirmos num acervo e não numa fila). Este algoritmo é muito semelhante ao algoritmo anterior, utilizando da mesma forma o acervo.

A principal diferença é que não há alterações dos tipos de rotas. Deste modo, colocando o tipo de rotas sempre a 1 é possível utilizar o mesmo algoritmo que o anterior, com as pequenas alterações visíveis no algoritmo 3. De notar que a variável *succs* é a união do conjunto de *peers*, *provider* e *clients*. Sobre a complexidade tal como anteriormente é de O((E + V) log V), sendo que o termo O(E log V) tem um maior peso que anteriormente, uma vez que todas as arestas são analisadas (mesmo não respeitando o protocolo BGP). Tal verifica-se, visto que a execução deste algoritmo é mais lenta.

Algorithm 3 MinSize

```
1:  procedure MinSize(graph,AS)
2:      Insert(AS, 1, 0)
3:      while RemovePrior(&AS_t, &type, &size) do
4:          if graph[AS_t].visit != AS then
5:              graph[AS_t].visit ← AS
6:              update_vector_shortest(size)
7:              for aux in all AS_t succs do
8:                  if graph[aux->AS].visit = AS then
9:                      continue
10:                 Insert(aux- > AS, 1, size + 1)
```

5 Resultados obtidos

A partir da rede do professor foi executado o algoritmo (previamente explicado), que avalia o tipo de rota elegida, para todas as possíveis ligações. Deste modo, foi possível saber como é composto a Internet que foi fornecida no ficheiro de texto e, com isso, produzir as estatísticas apresentadas na tabela 2.

Tabela 2: Divisão dos tipos de rotas.

Tipo de rota	Percentagem [%]
Client	0.61
Peer	11.24
Provider	88.15

É importante notar que para a realização das estatísticas, foram consideradas as rotas de tamanho zero, ou seja, de cada *AS* para a própria *AS* e foi considerado que eram rotas do tipo *client*.

O segundo algoritmo tem como função o cálculo do tamanho de rotas e, tal como o anterior, foi executado para todas as possíveis rotas entre *ASes* e, com os diferentes resultados obtidos para rotas comerciais e rotas não comerciais (rotas mínimas), tendo sido obtido o gráfico da figura 1. Verifica-se tal como esperado que a maioria das rotas são do tipo *provider*, tal deve-se ao número elevado de *stubs*.

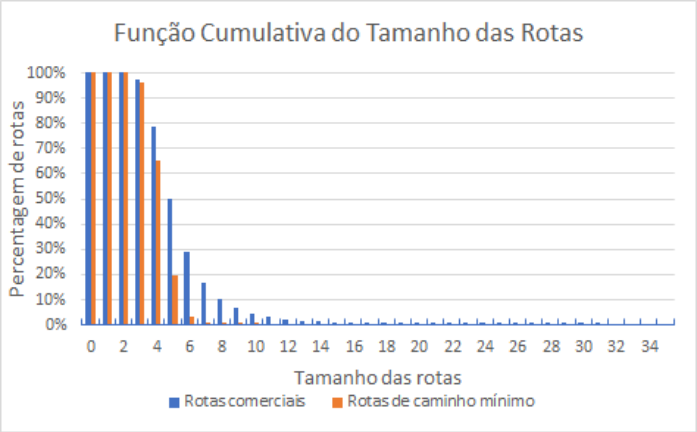

Figura 1: Gráfico da função cumulativa do tamanho das rotas para as rotas comerciais e não comerciais. De notar que os resultados são em percentagem sendo esta o quociente do número total de rotas com pelo menos tamanho x sobre o número total de rotas.

Analisando o gráfico, rapidamente se percebe que a percentagem de rotas de caminho mínimo se aproxima mais rapidamente de zero face às rotas comerciais, ou seja, as rotas de caminho mínimo têm sempre um tamanho menor, ou igual que a respectiva rota comercial. Isto deve-se ao facto de existirem políticas de roteamento entre *ASes*, facto que leva a um fenómeno conhecido por *path inflation*, que não existe nos caminhos mínimos.

Pode-se ainda constatar o facto de que a maior rota comercial tem um tamanho de 35, enquanto para o caso de caminhos mínimos tem-se um valor substancialmente menor, 11.

6 Conclusões

Relativamente à Internet fornecida, é notável reparar que embora contenha cerca de 40.000 *ASes*, qualquer rota entre duas quaisquer *Ases*, tem no máximo tamanho 35 para rotas comerciais e 11 para rotas não comerciais. Tal deve-se à Internet fornecida ser densa e cada *AS* ter um elevado número de ligações.

Em termos de optimização, é possível melhorar a performance dos algoritmos se em vez do acervo binário implementado, se utilizasse, por exemplo, um acervo de *Fibonacci*, e visto que no segundo algoritmo não é necessário respeitar o protocolo *BGP*, o acervo utilizado poderia ser substituído por uma fila, dado que o algoritmo de *Dijkstra* para grafos sem pesos é idêntico a uma *BFS*.

Em suma, pode-se afirmar que foi possível não só por em prática diversos algoritmos aprendidos nas aulas teóricas de ADRC, como também ficar com uma ideia geral e conhecedora do funcionamento das redes de Internet e da algoritmia por detrás do roteamento na mesma.