TÉCNICO LISBOA

1º Semestre 2021/2022

Mestrado em Engenharia Eletrotécnica e de Computadores

Algoritmia em Redes e Aplicações

Relatório do projeto

Nº 93148 Miguel Quinteiro Ribeiro miguel.quinteiro.ribeiro@tecnico.ulisboa.pt
Nº 93906 Diogo José Pereira Araújo diogoparaujo@tecnico.ulisboa.pt

Docente responsável: Prof. João Luís Costa Campos Gonçalves Sobrinho

Introdução

No âmbito da unidade curricular de Algoritmia em Redes e Aplicações, foi pedido para implementar um simulador e um algoritmo de forma a obter o estado estável do protocolo BGP para qualquer grafo, tendo em conta as restrições comerciais que caracterizam este protocolo e o custo de cada ligação.

Simulador

Um simulador é um calendário de eventos ordenados, capaz de simular as trocas de mensagens entre ASes, tendo em conta: instante de receção de cada mensagem, atrasos na receção das mensagens, filas de espera de cada conexão e relações comerciais entre as ASes.

De forma a implementar o simulador, foi necessário tomar as seguintes decisões iniciais. Primeiro, representar o grafo por uma lista de adjacências. Segundo, cada nó do grafo tem que ter uma tabela de encaminhamento, guardando todas as estimativas para um dado destino, a partir de todos os seus vizinhos, sendo necessário eleger a melhor. As mensagens trocadas entre ASes são interpretadas como eventos discretos no tempo que compõe o calendário.

De forma a implementar um calendário de eventos ordenados, o grupo recorreu a uma lista ordenada por ordem crescente do tempo de processamento de cada evento. Cada evento representa o anúncio que foi transmitido de um certo nó u para um dos seus vizinhos, sempre que este melhora a sua estimativa para um dado destino. Cada evento tem que conter o nó que anuncia a sua estimativa para um dado destino, o nó a que se destina o anúncio, o instante de tempo a que o evento deve ser processado (A_n) e a respetiva mensagem, que contém o nó que anunciou, o dado destino e o respetivo custo desse nó até ao destino.

Tendo definido quais as estruturas de dados que foram necessárias utilizar. Tem que se definir como inicializar, executar e terminar o calendário.

A inicialização do calendário é feita pelo seguinte evento externo: o destino anuncia a todos os seus vizinhos que consegue chegar a ele próprio com custo zero. Este evento externo, vai gerar tantos eventos quantos o número de vizinhos do destino que se anunciou.

A execução do simulador, consta no processamento do evento que está no calendário. O processamento de um evento consiste em: primeiro, atualizar a tabela de encaminhamento do nó a que se destina o anúncio, dado a mensagem que recebeu do vizinho que a anunciou. Observe-se o seguinte exemplo demonstrativo da atualização de tabela de encaminhamento de um dado nó v, que recebe um anúncio de um vizinho u, com uma dada estimativa para o nó destino t.

- Procurar t na tabela de encaminhamento de v
  - Se não encontrar t:
    - Criar o destino t
    - Criar o vizinho u
    - Tornar o caminho por u a melhor estimativa de v para o destino t.
  - Se encontrar t:
    - Procurar o vizinho u na tabela de encaminhamento de v, referente ao destino t
      - Se não encontrar: Criar o vizinho u e inseri-lo ordenadamente na respetiva tabela de encaminhamento
      - Se encontrar: Atualizar a estimativa do vizinho u e reordenar por ordem crescente de custo a respetiva tabela de encaminhamento
  - Atualizar a estimativa do custo de v relativamente ao destino t

Se a estimativa para um dado destino alterar, então o nó tem que anunciar essa nova estimativa para os seus vizinhos. Desta forma criam-se novos eventos, que têm que ser inseridos ordenadamente no calendário, para serem posteriormente processados.

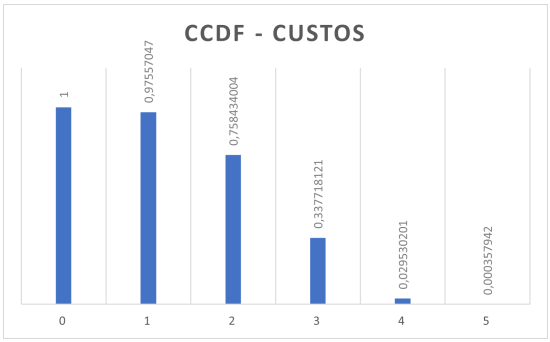
O simulador termina, quando já não houver mais eventos para processar no calendário.

É relevante referir mais alguns pontos. Um nó só anuncia a sua estimativa para um dado destino se, e só se, a ligação entre ele e o seu vizinho cumprir as restrições comerciais do BGP.

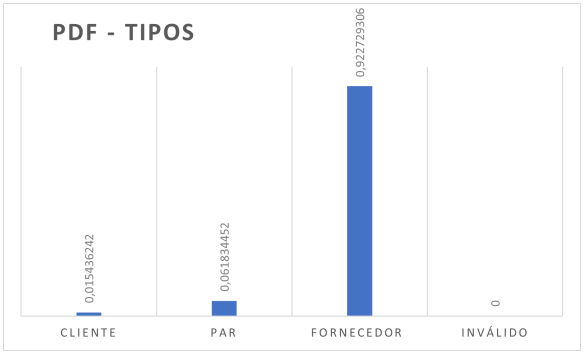
De forma a presevar a topologia FIFO de cada ligação, foi necessário implementar uma condição que garantisse que um novo evento referente a essa ligação não fosse processado antes do que um evento (também referente a essa ligação) que já tivesse sido criado à mais tempo. Não sendo necessário implementar uma fila de espera em cada ligação.

Estatísticas

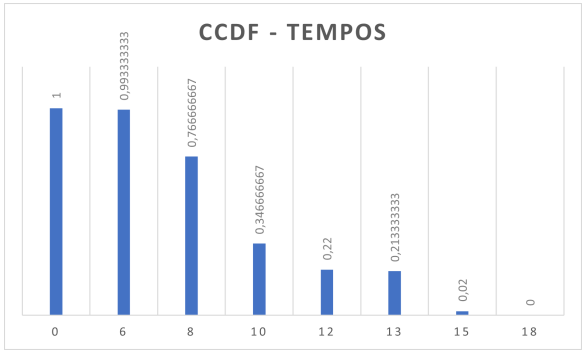
A estatísticas obtidas pelo algoritmo para o ficheiro "*Mini_Internet.tsv*"encontram-se na figura 1.



(a)



(b)



(c)

Figura 1: Estatísticas obtidas pelo simulador: (a) CCDF: custos (b) PDF: tipos (c) CCDF: tempos.

Algoritmo

O algoritmo implementado consiste numa versão modificada do *Reverse Generalized Bellman-Ford Algorithm*, com algumas subtilezas nada intuitivas.

O algoritmo recebe como parâmetros de entrada, a topologia do grafo e o nó de destino.

Primeiro, inicializa-se o algoritmo, percorrendo todos nós do grafo, criando uma tabela de encaminhamento para cada um destes e inicializando todos os parâmetros (*type*, *cost* e *parent*) como inválidos. Sendo que o nó que corresponde ao nó de destino será inicializado com todos os parâmetros a zero, visto que o destino é ele próprio.

Pelo *Reverse Bellman-Ford Algorithm*, o próximo passo consistiria em colocar o destino numa pilha FIFO. Depois, retirar esse nó da pilha, fazer a relaxação das suas ligações e inserir os respetivos vizinhos na pilha. No entanto, neste projeto é necessário ser mais cuidadoso, pois, é necessário ter em consideração dois atributos: tipo de cada rota e custo de cada rota. Tentar resolver este problema com apenas uma pilha iria tornar o algoritmo extremamente ineficiente. Portanto, decidiu-se implementar três pilhas, cada uma referente aos tipos de ligações possíveis: fornecedor-cliente; par-par e cliente-fornecedor.

Tendo em conta as relações comerciais, é necessário definir uma relação de prioridade entre as três pilhas. Ou seja, só se pode remover nós da pilha cliente-fornecedor se as outras duas estiveram vazias. Só se pode

remover elementos da pilha par-par se a pilha fornecedor-cliente estiver vazia. Portanto, primeiro é necessário remover todos os elementos da pilha fornecedor-cliente. Por uma questão de simplicidade é nesta pilha onde se insere o nó de destino, para depois fazer a relaxação das suas ligações. Desta forma, obtém-se uma complexidade de $\mathcal{O}(1)$ na determinação do tipo de rota, de todos os nós para um dado destino.

Também é necessário minimizar a complexidade da determinação do custo de cada rota. Para isso, é necessário ter em conta qual deverá ser a topologia das pilhas a usar. Decidiu-se usar uma topologia FIFO para cada uma das três pilhas. Para as pilhas fornecedor-cliente e par-par é fácil perceber que, com uma topologia FIFO, estas vão estar ordenadas por ordem crescente de custos. Isto deve-se ao facto de, nestas duas pilhas, apenas podermos inserir nós, cujos *parents* são provenientes da pilha fornecedor-cliente, logo um nó que seja inserido na pilha terá sempre um custo maior (estimativa pior) que os que já lá estavam, ficando as pilhas ordenadas por ordem crescente de custos. No entanto, isto não acontece para a pilha cliente-fornecedor. Pois, nesta última podem ser inseridos nós cujos *parents* foram retirados de qualquer uma das três pilhas. Logo, com uma topologia FIFO, a pilha não fica ordenada de forma crescente de custos. Isto poderia ser um problema, se se considerasse que o nó retirado da pilha está terminado[(i)], mas a subtileza é não considerar que o nó que foi retirado da pilha está terminado, e em fazer uma validação do nó que se acabou de retirar da pilha.

Antes de perceber a condição de validação, é crucial perceber como funciona o processo de relaxação. Ao remover um nó da pilha, é necessário fazer a relaxação das suas ligações, desde que essas sejam válidas de acordo com as restrições comerciais. Se isso assim for, então é necessário verificar se a estimativa de um dado nó ao destino melhora ou não. Se melhorar vai-se atualizar a tabela de encaminhamento desse nó, e inserir esse nó na sua respetiva pilha. A relaxação é feita a partir dos seguintes critérios:[(ii)]

- Ordem total: $(\mathrm{x},l_1) \preceq_{ToRc} (\mathrm{y},l_2)$, se $\mathrm{x} \preceq_{ToR} \mathrm{y}$ [(iii)] $\vee\ (x =_{ToR} \mathrm{y} \wedge\ \mathrm{l}_1 < l_2)$
- Operação de extensão: $(\mathrm{x},l_1) \oplus_{ToRc} (\mathrm{y},l_2) = \begin{cases} (x,\ \mathrm{l}_1 + l_2) & \text{Se x=P} \vee \text{y=C} \\ \bullet & \text{Se x}\neq\text{P} \vee \text{y}\neq\text{C} \end{cases}$

É importante perceber que o algoritmo usa duas estruturas para caracterizar a estimativa do nó para um dado destino: a tabela de encaminhamento, e as estimativas que são inseridas na pilha. A validação é feita após a remoção do nó da pilha. A estimativas que são retiradas da pilha, que podem ou não estar corretas, vão ser comparadas com o que o nó tem na sua tabela de encaminhamento, que está sempre atualizado. Se as estimativas diferirem, então este elemento é descartado, e remove-se o próximo nó da pilha. Se coincidirem, faz-se as respetivas relaxações.

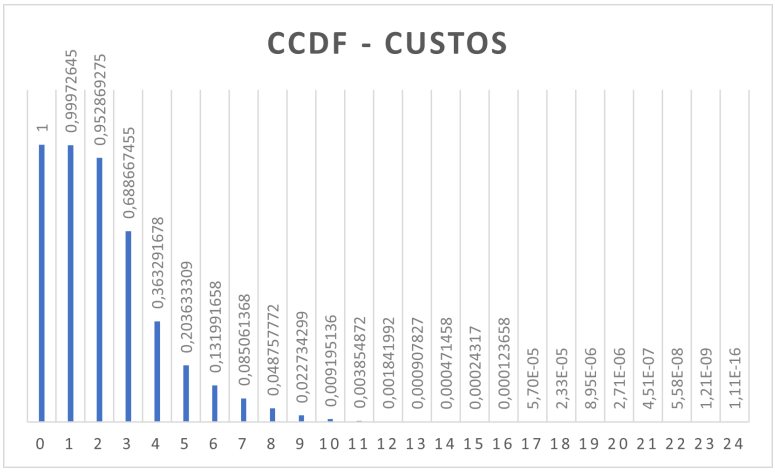
Esta solução causa um pequeno contratempo: temporariamente haverá nós que têm estimativas erradas para um dado destino, e isso vai se refletir na relaxação das suas ligações. No entanto, o algoritmo prova-se correto devido à condição de validação, e pelo facto que se definiu uma ordem de prioridade entre as três pilhas. Esta ordem de prioridade, força que cada nó respeite as regras de seleção do tipo de rota. Ou seja, sim é verdade que temporariamente os nós vão ter estimativas erradas, e será necessário fazer relaxações a mais, de forma a corrigir esses erros. No entanto, este é um pequeno preço a pagar, visto que desta forma, a complexidade de inserção e remoção nas três pilhas é $\mathcal{O}(1)$.

Desta forma o algoritmo apresenta uma complexidade $\mathcal{O}(m)$ [(iv)] para um dado destino, tornando o algoritmo extremamente eficiente. O algoritmo termina, qaundo todas as pilhas tiverem vazias.

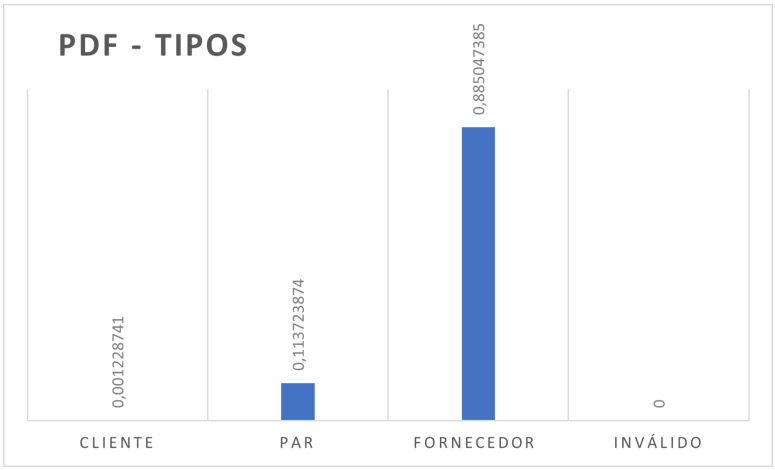
Estatísticas

A estatísticas obtidas pelo algoritmo para o ficheiro "*Complete_Internet.tsv*"encontram-se na figura 2.

[(i)] Um nó considera-se terminado quando já encontrou a melhor estimativa para um dado destino.
[(ii)] Nota: (tipo de rota, custo do caminho).
[(iii)] $\mathrm{C} \preceq_{ToR} \mathrm{R} \preceq_{ToR} \mathrm{P} \preceq_{ToR} \bullet$
[(iv)] m corresponde ao número de ligações do grafo.



(a)

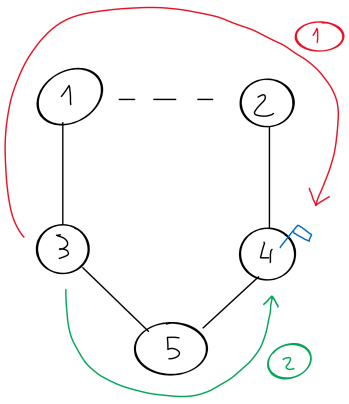


(b)

Figura 2: Estatísticas obtidas pelo algoritmo: (a) CCDF: custos (b) PDF: tipos.

Comparação entre as rotas escolhidas pelo BGP e os caminhos mais curtos

Devido às restrições comerciais, nem sempre as rotas escolhidas pelo BGP coincidem com o caminho mais curto de um nó para um dado destino. Tendo em conta o grafo representado à direita. Assumindo como o destino o nó 4. O nó 3 tem duas opções para chegar ao destino: caminho (1) ou (2), representados na figura. No entanto, o caminho (2) é comercialmente inválido, pois o nó 5 não vai exportar o que recebeu do nó 3 (seu fornecedor) para o nó 4 (outro fornecedor). Logo, o único caminho comercialmente válido de 3 para 4 é a rota (1). Ora, este caminho tem custo três, enquanto o caminho (2), que é comercialmente inválido, tem custo dois. Portanto, confirma-se que as rotas escolhidas pelo protocolo BGP nem sempre coincidem com o caminho mais curto de um nó para um dado destino.



Conclusões

Concluiu-se que o simulador é bastnte importante no entendimento da evolução de redes de ASes e dos seus estados transitórios. O grupo notou também que a simulação é computacionalmente mais exigente que a execução do algoritmo implementado, na medida em que é necessário guardar em memória o estado da rede de ASes, bem como a troca de mensagens entre todos os vértices do grafo constituído pelas ASes. A simulação torna-se assim necessária para entender certas propriedades derivadas do carácter distribuído do BGP, contudo, não é eficiente para determinar caminhos ótimos entre as ASes. Para tal é necessário o algoritmo, muito mais eficiente que a simulação, mas que não tem em conta o carácter distribuído nem o regime transitório característicos do BGP.

Uma ontra conslusão importante de se mensionar foi perceber a diferença entre complexidade computacional quadrática e linear. Inicialmente, começou-se por implementar cada umas das três pilhas, como pilhas ordenas por ordem crescente do custo de cada rota. Isto implicava que se tinha uma complexidade de $\mathcal{O}(n)$ [(v)] na determinação do custo de cada rota, o que implicava que o algoritmo teria uma complexidade computacional quadrática $\mathcal{O}(m \cdot n)$ para um dado destino. Para redes de pequena dimensão, não houve problemas em termos do tempo de execução. No entanto, quando se testou o algoritmo com uma rede de grandes dimensões este não terminou (para todos os destinos). Tendo feito a alteração na topologia das pilhas para FIFO, garantiu-se uma complexidade computacional linear, como já foi demonstrado previamente. Com complexidade linear para um dado destino, o algoritmo termina a rede "*Complete_Internet.tsv*"em tempo útil (para todos os destinos).

[(v)] n corresponde ao número de nós do grafo.