

Algoritmos Distribuídos Probabilísticos

FELIPE VELLOSO ALVES

Monografia apresentada ao departamento de informática da Universidade Federal do Paraná para obtenção do grau de bacharel em ciência da computação.

Orientador: **Prof. Dr. Jair Donadelli Júnior**

Curitiba, 14 de novembro de 2006

Resumo

Serão apresentados nesta monografia problemas clássicos da computação distribuída e os resultados obtidos pelo uso de algoritmos probabilísticos no desenvolvimento de suas soluções. Os problemas apresentados são os mesmos presentes na seção 3 de Gupta et al. (1994), à exceção do problema do *Escalonamento das Guardas de Comunicação*. Decidiu-se não apresentar esse problema uma vez que é muito semelhante ao problema do *Jantar dos Filósofos*. O objetivo é fornecer uma descrição mais completa e detalhada do que as apresentadas naquela fonte.

Sumário

Capítulo 1. Introdução	7
1. Algoritmos Distribuídos	7
2. Algoritmos Probabilísticos	9
Capítulo 2. Problemas Clássicos	11
1. Introdução	11
2. O problema do Jantar dos Filósofos	11
3. O problema da Eleição de Líder	22
4. O problema de Roteamento em um n -cubo	31
5. O problema dos Generais Bizantinos	42
Referências Bibliográficas	51
Índice Remissivo	53
Apêndice A. Equações e teoremas	55
1. Equação para e^x	55
2. Convergência de Cauchy	55
3. Limite de Chernoff	55

CAPÍTULO 1

Introdução

1. Algoritmos Distribuídos

Como explicado em Lamport e Lynch (1990), em *computação distribuída* o termo *distribuída* significa espacialmente espalhado. Computação distribuída é aquela executada em um sistema espacialmente distribuído. Entretanto, tudo depende da perspectiva com que se encara o sistema. Um processador, que para um projetista de hardware é um sistema distribuído composto por múltiplos componentes individuais que se comunicam entre si, é uma entidade única para um programador que trabalhe com uma linguagem de alto nível.

O modelo que usaremos para representar computação distribuída aqui é o modelo de *processos*, em que a computação é realizada pela execução de um conjunto de processos concorrentes. *Algoritmos Distribuídos* são então os algoritmos executados por estes processos.

O modelo de processos inclui muitas variações, principalmente no que se refere ao meio de comunicação entre os processos. Usualmente, os processos comunicam-se através de trocas de mensagens, isto é, existe uma fila onde processos que estejam enviando uma mensagem a adicionam e de onde processos que estejam lendo uma mensagem a retiram. Também o funcionamento da fila é variável: pode-se ter filas a que todos os processos tenham acesso, filas específicas para cada processo, filas infinitas ou finitas, enfim, múltiplas variedades com que modelar problemas e soluções.

Outro método bastante utilizado para a comunicação entre processos é o das variáveis compartilhadas. Nesse modo de comunicação, há variáveis cujo acesso compartilhado permite que os processos influenciem o comportamento uns dos outros. Também aqui há uma variedade considerável de configurações, envolvendo limitações ao acesso de variáveis por processos específicos.

Um terceiro método de comunicação envolve primitivas de sincronização para o envio de mensagens. Neste modo, um processo que queira enviar uma mensagem aguarda que o destinatário esteja pronto para recebê-la, bem como um processo que queira receber uma mensagem espera até que o remetente a envie.

As principais medidas de complexidade para algoritmos distribuídos são o tempo e as mensagens enviadas. Para o tempo, podemos medir tanto o tempo total gasto com o envio de mensagens quanto o tempo total de computação. Entretanto, costuma-se calcular apenas o tempo de envio de mensagens, uma vez que atualmente o tempo utilizado no envio de dados entre processadores é da ordem de milissegundos, enquanto o tempo de processamento é da ordem de microssegundos. Tradicionalmente, mede-se a complexidade temporal através da quantidade total de ligações entre processadores atravessadas por todas as mensagens enviadas.

A medida mais comum para complexidade de mensagens é a quantidade total de mensagens enviadas. Logicamente, busca-se resolver o problema com a menor quantidade de envio de mensagens possível. Nos casos em que o tamanho das mensagens varia com o número de processos, torna-se interessante medir também o número de bits utilizado por cada mensagem, em função do número total de processos.

Também em algoritmos distribuídos há interesse no estudo de limites inferiores para a complexidade de algoritmos, bem como as complexidades de pior caso e esperada para determinados algoritmos.

Neste trabalho serão apresentados problemas envolvendo os dois primeiros modos de comunicação citados: a comunicação por variáveis compartilhadas, utilizada no problema do *Jantar dos Filósofos*, e as filas de mensagens, usadas nos problemas de *Eleição de Líder*, *Roteamento em um n -cubo* e dos *Generais Bizantinos*. Para um problema envolvendo primitivas síncronas de comunicação, veja em Gupta et al. (1994), seção 3.2, o problema do *Escalonamento das Guardas de Comunicação*.

Os problemas relativos a sistemas de processos distribuídos envolvem usualmente a simetria entre os processos. Em alguns casos, como nos problemas do *Jantar dos Filósofos*, a

simetria impede qualquer solução. Em outros, como no caso do *Roteamento em um n-cubo*, a simetria é causa de problemas na eficiência do algoritmo.

Uma solução proposta para a quebra de simetria é o uso de algoritmos probabilísticos, introduzidos a seguir.

2. Algoritmos Probabilísticos

Algoritmos Probabilísticos são algoritmos que em fazem uso de bits aleatórios em sua computação, de maneira que seu comportamento pode variar para uma mesma entrada fixa. Este funcionamento pode ser modelado de várias maneiras como, por exemplo, o arremessar de uma moeda (justa ou viciada), o sorteio de um valor contido em um conjunto ou como uma entrada adicional composta por uma seqüência de números aleatórios.

O interesse pelo estudo de algoritmos probabilísticos possui uma série de motivos. Há problemas em que o uso de decisões aleatórias nos fornece um algoritmo cujo tempo de execução esperado é significativamente inferior ao da melhor solução determinística conhecida. Por exemplo, no *Roteamento de Pacotes em um n-cubo*, existe um algoritmo probabilístico com complexidade $O(n)$, sendo que para soluções determinísticas a complexidade é $\Omega(\sqrt{2^n})$. Há casos em que pode-se provar que é impossível desenvolver um algoritmo determinístico para resolver um determinado problema que possui uma solução probabilística, como no clássico problema do *Jantar dos Filósofos*. Outra vantagem importante reside no fato de que geralmente a versão probabilística da solução de um problema um problema é mais fácil de se entender e implementar do que seu par determinístico. Como exemplo, pode-se citar o *Teste de Primalidade* (veja Gupta et al. (1994), seção 2.2), cuja solução aleatória é mais simples e mais rápida, ainda que a solução determinística tenha complexidade polinomial.

Existe entretanto, analogamente ao caso da troca entre espaço e tempo em algoritmos determinísticos, uma troca de recursos necessários e certeza da solução. Para desfrutar das vantagens da aleatoriedade, sacrifica-se a noção da garantia absoluta de uma solução correta e assume-se que o algoritmo fornece solução correta com probabilidade $1 - \varepsilon$. Apesar disso, como se verá adiante, existem algoritmos para os quais ε é zero, de maneira que o algoritmo sempre termina fornecendo uma solução correta, como no problema dos *Generais Bizantinos*.

Tal troca nos leva à classificação dos algoritmos probabilísticos em algoritmos *Monte Carlo* e *Las Vegas*. Algoritmos do tipo Monte Carlo sempre fornecem uma resposta, provavelmente correta. Algoritmos do tipo Las Vegas podem não fornecer resposta alguma, mas quando o fazem ela é certamente correta. Para a classe *Las Vegas* pode-se citar como exemplo o *Quicksort probabilístico*, cuja resposta é sempre correta e o algoritmo é provavelmente eficiente.

É importante ressaltar que a *análise probabilística de algoritmos* (também chamada de *análise de caso médio*) é distinta do uso da aleatoriedade em algoritmos. Enquanto algoritmos probabilísticos são uma classe de algoritmos, a análise probabilística é o estudo do comportamento de um algoritmo, não necessariamente probabilístico, quando sua entrada é tomada de um conjunto de instâncias segundo uma distribuição de probabilidade.

CAPÍTULO 2

Problemas Clássicos

1. Introdução

A seguir, serão apresentados os problemas clássicos da computação distribuída estudados, na seguinte ordem:

- (1) O problema do Jantar dos Filósofos,
- (2) O problema da Eleição de Líder,
- (3) O problema do Roteamento em um n -cubo e
- (4) O problema dos Generais Bizantinos.

Para cada problema veremos seu enunciado, as restrições que desejamos para sua solução, algumas limitações encontradas quando se tenta resolvê-lo com algoritmos determinísticos e uma solução probabilística proposta, bem como uma conclusão a respeito dos resultados obtidos pela solução descrita.

2. O problema do Jantar dos Filósofos

2.1. Introdução. Nesta seção, será apresentado o problema do *Jantar dos Filósofos*. Inicialmente, o problema será enunciado informalmente, para a seguir se descrever sua representação computacional e as restrições de uma solução. Adiante, será exibida a prova de que não há solução determinística realmente distribuída para tal problema. Por fim, exibirei a solução probabilística apresentada em Lehmann e Rabin (1981), bem como a prova de que resolve o problema satisfazendo todas as suas restrições com probabilidade 1.

O Jantar dos Filósofos é um dos problemas mais fundamentais da computação distribuída. Originalmente proposto em Dijkstra (1971), é uma ilustração de um problema de alocação de recursos em sistemas distribuídos. O problema segue como consta em Lehmann e Rabin (1981).

PROBLEMA DO JANTAR DOS FILÓSOFOS. *Uma série de filósofos está reunida para um jantar. A vida de um filósofo consiste basicamente em pensar. Enquanto está pensando, um filósofo não interage com os outros filósofos. Entretanto, o filósofo acaba por sentir fome em algum momento de sua vida. Para se alimentar, ele dispõe de um prato de macarrão que nunca se esvazia. O macarrão encontra-se de tal forma oleoso que é impossível comê-lo com apenas um garfo, sendo necessários dois para tal. Cada filósofo possui um garfo do lado esquerdo e um do lado direito de seu prato, sendo que entre dois filósofos encontra-se apenas um garfo — portanto o número de garfos é igual ao número de filósofos. Um filósofo pode pegar apenas um garfo de cada vez. É obviamente impossível utilizar um garfo que esteja sendo utilizado por um vizinho. Uma vez que um filósofo tenha dois garfos em suas mãos, ele alimenta-se até estar satisfeito, devolve os dois garfos e torna a pensar. Um filósofo faminto que não seja capaz de pegar seus dois garfos para todas as vezes que tente — o garfo sempre está em posse de seu vizinho, quando o filósofo tenta pegá-lo — entra em inanição.*

De maneira mais concisa, um filósofo opera indefinidamente no ciclo: pensar, tentar, comer. Para comer um filósofo necessita de acesso exclusivo a dois recursos, cada um compartilhado com um vizinho.

O problema computacional consiste em projetar um protocolo que represente os filósofos e os garfos de maneira apropriada, para que se comportem como as entidades descritas no enunciado. O protocolo deve garantir que os filósofos comam.

2.2. Restrições e propriedades da solução. A seguir são apresentados algumas restrições para soluções para esse problema. Isto se justifica por motivos estéticos, bem como práticos.

A primeira restrição é o da solução distribuída. São interessantes apenas soluções em que não há um processo controlador ou uma memória central a que todos os outros processos tenham acesso. Os únicos agentes devem ser os processos que representam os filósofos, sendo cada garfo modelado por uma célula de memória acessível apenas aos processos que representam os filósofos que o compartilham.

A segunda restrição é a de simetria: todos os processos devem ser idênticos. É bastante natural pensar em um sistema composto por uma série de pequenos processos simples. Na verdade, tão simples que seriam mesmo incapazes de manter algum identificador individual. Estamos interessados, portanto, em soluções em que o estado inicial de cada processo, bem como o valor inicial de cada variável compartilhada é exatamente o mesmo. Isso se dá pois uma solução que não respeitasse tal restrição poderia facilmente programar protocolos distintos para cada filósofo, fugindo da proposta inicial do problema.

O objetivo é encontrar um protocolo que, respeitando as restrições acima, garanta que os filósofos se alimentem. Definiremos dois graus de eficácia para tais soluções. Caso o protocolo garanta que todo filósofo faminto venha a comer, diz-se que ele é *livre de travamento*. Uma noção mais fraca é a de que caso um filósofo sinta fome — por exemplo, Platão —, **algum** filósofo se alimentará com certeza (não necessariamente Platão). Tal classe de protocolos é chamada de *livre de “deadlock”*.

Quanto ao processo de escalonamento, não há nenhum tipo de hipótese em relação às taxas de atividade de cada processo. Não está excluída, portanto, a possibilidade de que o escalonador seja malicioso e trabalhe contra a solução, fazendo o máximo possível para impedir que os filósofos se alimentem. O escalonador pode até mesmo ser capaz de ler a memória compartilhada e privada de todos os processos. Considerando o modelo usual para algoritmos probabilísticos, em que os bits aleatórios são lidos seqüencialmente pelo algoritmo, o escalonador pode acessar apenas os bits já lidos, sendo incapaz de “prever” o resultado de leituras futuras. Como se afirma em Lehmann e Rabin (1981), essas suposições demonstram de forma bastante clara a força do algoritmo proposto. Isso ressalta um dos princípios fundamentais do desenvolvimento de sistemas: supôr que o pior caso é inevitável.

Cada uma das variáveis representando os garfos é compartilhada por um par de filósofos, ou processos. Presume-se que o problema de acesso às variáveis está previamente resolvido, ou seja, nunca dois processos acessam a mesma variável ao mesmo tempo e sempre que um processo requisita acesso a uma variável compartilhada, ele acabará por recebê-lo. Pode-se imaginar que o acesso à variável toma um tempo tão pequeno quando comparado ao tempo

em que um processo está ativo que conflitos de acesso não ocorrem, ou que são tratados pelo hardware quando acontecem.

Isso nos garante que um filósofo pode verificar se um garfo está disponível e pegá-lo sem que seja incomodado por algum de seus vizinhos. Como se explicará adiante, pegar e devolver um garfo são mudanças no valor de uma variável compartilhada.

2.3. Não-existência de solução determinística. Pode-se provar facilmente que não existe protocolo que resolva o problema do Jantar dos Filósofos respeitando as restrições estabelecidas, em que o comportamento dos filósofos seja representado por um processo determinístico.

Definimos por *unidade atômica de tempo* o espaço de tempo necessário para que um processo execute exatamente uma instrução.

TEOREMA 1. *Não existe uma solução livre de deadlock, distribuída e simétrica para o problema do Jantar dos Filósofos.*

DEMONSTRAÇÃO. Suponha que exista uma solução livre de deadlock, distribuída e simétrica para o problema do Jantar dos filósofos. Vamos definir um escalonador que impeça a todos os filósofos de se alimentar, ou seja, vamos mostrar que não há solução livre de deadlock. Sem ferir nenhuma das hipóteses, podemos nomear os processos de 1 a n . Basta que o escalonador ative cada um dos processos por uma unidade atômica de tempo ordenadamente, de 1 a n , e repita esta ordem de ativação indefinidamente. Considerando-se que os processos se encontram inicialmente no mesmo estado, que com tal escalonador a simetria se preserva a cada série de n ativações e que é impossível que todos os filósofos estejam se alimentando simultaneamente, temos um escalonador que impede que todos os filósofos se alimentem, ou seja, que existe um escalonador que garante o deadlock para o algoritmo inicial. Por contradição, temos que não existe uma solução livre de deadlock, realmente distribuída e simétrica para o problema do Jantar dos Filósofos. \square

2.4. Solução probabilística. Como se pode ver, o que impede a solução determinística é a simetria entre os processos. Portanto, se quisermos prover uma solução para o problema,

devemos encontrar uma maneira de quebrar essa simetria. Para tal, vamos equipar os filósofos com o livre-arbítrio, permitindo que escolham aleatoriamente qual dos dois garfos tentarão pegar, de maneira que as leis da probabilidade garantam que, com probabilidade um, a simetria seja quebrada.

O algoritmo a seguir, chamado de *Algoritmo dos Filósofos Livres*, proporciona tal capacidade para os filósofos. A função R retorna o complemento do argumento no conjunto $\{Esquerdo, Direito\}$.

Algoritmo 1 Filósofos Livres

```

1: enquanto VERDADEIRO
2:   pensar;
3:   tentando := VERDADEIRO;
4:   enquanto tentando
5:     sortear aleatoriamente um elemento L do conjunto  $\{Esquerdo, Direito\}$ ; {Com pro-
        babilidade uniforme}
6:     esperar garfo L ficar disponível;
7:     pegar garfo L;
8:     se garfo R(L) está disponível
9:       pegar garfo R(L);
10:    tentando := FALSO;
11:   senão
12:     devolver garfo L;
13:   comer;
14:   devolver garfos; {Um de cada vez, em ordem arbitrária}

```

Definição. Uma *ação atômica* é a execução de exatamente uma instrução qualquer de um algoritmo.

Definição. Um *escalonador* é uma função que, baseada em todo o comportamento dos processos P_i , $1 \leq i \leq n$, define qual o próximo processo P_i a ser ativado, isto é, o próximo

processo a efetuar uma ação atômica. Por todo o comportamento dos processos entenda-se “todas as ações atômicas e resultados de sorteios aleatórios até o presente momento”.

Tal definição nos dá um escalonador que não simplesmente nos fornece o próximo processo a ser ativado baseado na seqüência de ativações, mas em todo o passado dos processos, incluindo seu funcionamento interno e resultados de sorteios aleatórios. Esse escalonador é obviamente muito poderoso e pode sem muita dificuldade definir uma seqüência de ativações visando o fracasso do algoritmo do protocolo.

Para um escalonador específico S e para uma seqüência de sorteios D (onde D é uma seqüência infinita de valores do conjunto $\{Esquerdo, Direito\}$), temos uma *computação* $C = COM(S, D)$, que é uma seqüência infinita $(A, P_i)_t$ de ações atômicas, onde A é a ação atômica efetuada, P_i é o processo que a executou e t é o instante em que a ação se passou. É importante notar que uma computação é interminável e demonstra a vida completa do sistema. Chamaremos de *computação finita* uma seqüência finita de ações atômicas. O i -ésimo elemento da computação C é a ação atômica que se passa na unidade de tempo i . Presume-se que nunca duas ações aconteçam exatamente ao mesmo tempo, mas tal restrição pode ser desconsiderada se mantivermos que dois ou mais acessos à uma mesma célula memória compartilhada nunca ocorrem conjuntamente.

Definição. Uma computação C é *justa* se em C todos os processos são ativados um número infinito de vezes. Um escalonador S é *justo* se para todas as seqüências infinitas de sorteios D a computação $COM(S, D)$ é justa.

A partir daí temos que, se um escalonador S é justo, então para qualquer computação $C = COM(S, D)$ nenhum processo é interrompido.

Pressupomos que no espaço de probabilidade de todas as seqüências infinitas de sorteios D a distribuição é uniforme. A função COM então associa a um escalonador S uma distribuição de probabilidade no espaço de todas as computações. A probabilidade de um conjunto E de computações sendo definida como a probabilidade do conjunto de seqüências de sorteios D tais que $COM(S, D)$ está em E .

A única pressuposição com relação ao escalonador é a de que este é justo. O teorema 6, enunciado na página 21, então demonstra que certas propriedades valem para **todo** escalonador justo.

O objetivo é demonstrar que, no sistema dos filósofos livres, a probabilidade de ocorrência de *deadlock* é zero. Primeiramente serão definidos os eventos em questão.

Definição. Uma computação em *deadlock* é uma computação em que existe um momento t , no qual um filósofo está tentando comer, mas a partir do qual nenhum filósofo come. Um filósofo P_i está *trancado* (ou em *inanição*) em uma computação C se existe um momento t em que P_i está tentando comer, mas após o qual P_i não come.

Para um escalonador justo S , existe uma probabilidade bem definida para que uma computação $C = COM(S, D)$ esteja em *deadlock*. Como definido anteriormente, a probabilidade de um conjunto E de computações para um escalonador específico S é a probabilidade do conjunto de seqüências de sorteios D_i tais que para todo i vale que $COM(S, D_i) \in E$. Para uma computação específica $C_j = COM(S, D_j)$, temos que $\mathbb{P}(\{C_j\}) = \mathbb{P}(\{D_j\})$. Isso se verifica porque, para um escalonador S e uma seqüência de sorteios D específicos, temos apenas uma computação $C = COM(S, D)$, uma vez que COM é uma função. Assim sendo, se fixarmos S , temos que D define se a computação está ou não em *deadlock*.

Vamos denotar por $DL(S) = \mathbb{P}(\{D: COM(S, D) \text{ está em } \textit{deadlock}\})$. A intenção é provar que $DL(S) = 0$ para todo escalonador justo S .

Para tanto, faremos uso de quatro lemas descritos a seguir. Estes lemas se referem genericamente a dois filósofos: Platão e seu vizinho à direita, Aristóteles.

LEMA 2. *Se Platão pega algum garfo um número infinito de vezes mas Aristóteles pega um garfo apenas um número finito de vezes, então, com probabilidade um, Platão se alimenta um número infinito de vezes.*

DEMONSTRAÇÃO. Se Aristóteles pega um garfo apenas um número finito de vezes, podemos afirmar que existe um instante t a partir do qual Aristóteles nunca pega um garfo. Logo, temos que a partir de t o garfo compartilhado por Platão e Aristóteles só será levantado por Platão. Mais do que isso, pode-se garantir que a partir de t Platão sempre pega o garfo

compartilhado com Aristóteles quando tenta fazê-lo. Assim, caso Platão sorteie o garfo esquerdo — executando a linha 5 — ele certamente se alimentará, pois esperará até seu vizinho esquerdo soltar o garfo e será capaz de pegar tanto o garfo esquerdo quanto o direito. O caso em que Platão espera infinitamente por seu garfo esquerdo está eliminado pela hipótese, pois isto implicaria que Platão pega um garfo um número finito de vezes. Caso Platão sorteie o garfo direito, ele pode ou não se alimentar, dependendo da disponibilidade do garfo esquerdo. Agora basta mostrar que, com probabilidade um, Platão sorteia o garfo esquerdo infinitas vezes e portanto alimenta-se infinitas vezes também. Imagine que Platão sorteia o garfo esquerdo um número finito de vezes. Podemos enumerar os casos em que isso acontece, identificando cada caso pela sequência de sorteios efetuada por Platão até a última vez que sorteia seu garfo esquerdo. Ou seja, temos um conjunto infinito e enumerável $F = \{D: \text{Platão sorteia seu garfo esquerdo um número finito de vezes em } COM(S, D)\}$. Temos que o conjunto Ω de todas as possíveis seqüências de sorteios D é infinito e não-enumeráveis. Considerando o espaço de probabilidade que nos interessa, em que todos os elementos de Ω estão distribuídos uniformemente, temos que $\mathbb{P}(F) = 0$, donde concluímos que, com probabilidade um, Platão sorteia o garfo esquerdo infinitas vezes e, portanto, se alimenta infinitas vezes. \square

LEMA 3. *Em uma computação em deadlock, todos os filósofos pegam algum garfo um número infinito de vezes, com probabilidade um.*

DEMONSTRAÇÃO. Vamos provar este lema por contradição. Suponha que exista uma computação C que esteja em deadlock e em que pelo menos um filósofo pega um garfo apenas um número finito de vezes. Por praticidade, chamemos um filósofo que pega um garfo um número finito de vezes um filósofo *impedido*. Sem que se perca a generalidade, seja Aristóteles um filósofo impedido. Caso Platão não seja um filósofo impedido, pelo lema 2 temos que, com probabilidade um, Platão se alimenta um número infinito de vezes, logo, C não está em deadlock. Caso Platão seja um filósofo impedido, mantém-se o raciocínio anterior, com Platão fazendo as vezes de Aristóteles. É fácil ver que a situação em que um filósofo impedido tem como vizinho à esquerda um filósofo não-impedido sempre ocorre quando existe pelo menos um filósofo impedido. O único caso em que isto não acontece é aquele em que todos os

filósofos se encontram impedidos, mas tal caso é impossível: se os dois vizinhos de um filósofo são impedidos, existe um momento t a partir do qual nenhum dos vizinhos do filósofo em questão pega um garfo. Logo, sempre que o filósofo tentar pegar um garfo após o momento t — qualquer que seja — ele o encontrará livre e o pegará. Como a computação é justa por hipótese, o filósofo será ativado infinitas vezes, de modo que pegará um garfo infinitas vezes e é um filósofo não-impedido. \square

Para cada instante temos uma configuração de sorteios aleatórios já feitos. Vamos chamar duas configurações A e uma posterior B de *disjuntas* caso, entre A e B todos os filósofos efetuaram pelo menos um sorteio aleatório.

LEMA 4. *Seja F uma computação finita composta por t passos tal que, no instante t , tanto Platão quanto Aristóteles estão tentando comer, o último sorteio aleatório de Platão foi Esquerdo e o último sorteio aleatório de Aristóteles foi Direito. Considere todas as computações infinitas C que sejam continuações de F . Então temos que pelo menos um dentre Platão e Aristóteles se alimenta antes da próxima configuração disjunta da atual em C com probabilidade um.*

DEMONSTRAÇÃO. Sabemos que se Platão e Aristóteles estão tentando comer, então cada um deles pode se encontrar em alguma dos seguintes estados:

- (a) O filósofo está esperando que o garfo sorteado seja disponibilizado, ou
- (b) O filósofo está em posse do garfo sorteado.

O estado (b) é alcançado por um filósofo que, estando no estado (a), encontra o garfo esperado disponível e o levanta.

Queremos provar que algum dos filósofos se alimenta antes que efetuar qualquer sorteio, ou seja, antes da próxima configuração disjunta da atual.

Vamos considerar inicialmente o caso em que tanto Platão quanto Aristóteles se encontram no estado (b). Podemos concluir que o próximo filósofo a ser ativado irá se alimentar antes de seu próximo sorteio pois encontrará o garfo comparilhado pelos dois disponível, o levantará e se alimentará.

Vamos agora considerar o caso em que, sem perda de generalidade, Platão se encontra no estado (b) e Aristóteles no estado (a). Se Platão for o próximo dentre os dois a ser ativado, ele encontrará o garfo compartilhado disponível, o levantará e se alimentará antes de ter feito algum sorteio. Se Aristóteles for ativado, ele pode tanto permanecer no estado (a) quanto progredir para o estado (b). Digamos que ele se mantenha no estado (a). Então podemos mantê-lo no caso atual. Se ele progredir para o estado (b), então temos que ambos os filósofos estão no estado (b) e chegamos ao caso anterior.

Finalmente, analisamos o caso em que ambos os filósofos se encontram no estado (a). Sendo ativado, qualquer um dos dois filósofos em questão se manterá no estado (a) ou avançará para o estado (b). Até que algum deles atinja o estado (b), mantêm-se o caso atual. No momento em que algum deles encontrar o garfo esperado disponível e o levantar, chegamos ao caso anterior. \square

LEMA 5. *Se todo filósofo pega um garfo um número infinito de vezes e, para um dado instante t , a configuração de sorteios aleatórios já efetuados é A , então o protocolo chegará, com probabilidade um, a uma configuração posterior B , disjunta de A , em que o último sorteio de algum filósofo foi Esquerdo e o último sorteio de seu vizinho à direita foi Direito.*

DEMONSTRAÇÃO. Os dois únicos casos em que a situação final do lema não ocorre são aqueles em que todos os filósofos obtiveram o mesmo valor no seu último sorteio; ou *Esquerdo* ou *Direito*. Como ambos os valores são sorteados sempre com probabilidade $\frac{1}{2}$, temos que com probabilidade $2\left(\frac{1}{2}\right)^p = \left(\frac{1}{2}\right)^{p-1} = \frac{1}{2^{p-1}}$, onde p é o número de filósofos, todos os filósofos sorteiam o mesmo valor em seu último sorteio imediatamente antes de uma configuração qualquer.

Seja $A_1, A_2, \dots, A_n, \dots$ uma seqüência infinita de configurações disjuntas de sorteios aleatórios prévios, onde A_n é a configuração disjunta imediatamente anterior a A_{n+1} , ou seja, pelo menos um filósofo fez exatamente um sorteio entre A_n e A_{n+1} , para todo n . Para um intervalo de configurações $A_n, A_{n+1}, \dots, A_{n+k}$, temos que a probabilidade de que todos os filósofos tenham sorteado o mesmo valor em todos os sorteios que antecederiam imediatamente algum estado A_i para $n < i \leq k$ é de $\left(\frac{1}{2^{p-1}}\right)^k$. Como estamos lidando com computações infinitas

e $\lim_{k \rightarrow \infty} \left(\frac{1}{2^{p-1}}\right)^k = 0$, a partir de qualquer configuração A surgirá, com probabilidade um, uma configuração disjunta B tal que, no último sorteio antes de B , algum filósofo sorteou *Esquerdo* e seu vizinho à direita sorteou *Direito*. \square

Finalmente, chegamos ao teorema que nos diz que a solução proposta para o problema é certamente livre de deadlock.

TEOREMA 6. *Para todo escalonador S justo, $DL(S) = 0$.*

DEMONSTRAÇÃO. Provaremos o teorema por contradição. Suponha que $DL(S) > 0$. Podemos então nos referir às probabilidades dos eventos relativos ao deadlock. Pelo lema 3, temos que, com probabilidade um (relativamente ao evento de uma computação em deadlock), todos os filósofos executam um número infinito de sorteios aleatórios. Pelo lema 5, acontecerá uma seqüência infinita de configurações disjuntas de sorteios aleatórios satisfazendo as hipóteses do lema 4, digamos $A_1, A_2, \dots, A_n, \dots$. Pelo lema 4, algum filósofo come entre A_n e A_{n+1} , para todo n , com probabilidade um. Chegamos então à conclusão de que, relativamente ao evento de uma computação em deadlock, computações livres de deadlock têm probabilidade um. Desta maneira, mostramos que a ocorrência de deadlock deve ter probabilidade zero. \square

2.5. A solução dos Filósofos Livres não é livre de travamento. Ainda que seja livre de deadlock, a solução proposta para o problema do Jantar dos Filósofos não é livre de *travamento*. É possível provar, inclusive, que existe um escalonador que faz $n - 1$ filósofos entrarem em inanição, usando o protocolo dos Filósofos Livres. Ainda em Lehmann e Rabin (1981), é proposta uma solução que evita a situação de travamento. Tal solução, chamada de *Algoritmo dos Filósofos Corteses*, inclui variáveis compartilhadas adicionais, usadas para que os filósofos tenham um comportamento “bem educado” em que um filósofo, ao sortear um garfo para pegar, verifica se foi o último a usá-lo. Se for o caso, o filósofo aguarda seu vizinho usá-lo para só então o levantar.

2.6. Conclusões. Como pudemos verificar, o uso de sorteios aleatórios se provou uma ferramenta bastante útil, nos dando a capacidade de resolver um problema que sem ela era

impossível de solucionar. A prova para os lemas dessa seção foram desenvolvidas pelo autor dessa monografia. As provas para os teoremas são baseadas em Lehmann e Rabin (1981).

No problema seguinte veremos novamente a ação incapacitante da simetria, e como o uso de algoritmos probabilísticos pode nos ajudar a superá-la.

3. O problema da Eleição de Líder

3.1. Introdução. Nesta seção, apresentarei o problema da *Eleição de Líder*. Inicialmente, será fornecida uma motivação para o estudo deste problema, bem como seu enunciado formal. Por fim, é descrita a solução proposta em Itai e Rodeh (1990) e os resultados em termos de complexidade associados a ela.

A Eleição de Líder é um problema cuja solução tem aplicação prática direta, pois a topologia de redes em anel é de fato utilizada, ainda que não seja a mais popular delas. Assim sendo, o problema não é uma representação geral de um tipo de problema como o Jantar dos Filósofos, que representa os problemas de compartilhamento de recursos, mas é um problema que pode ser verificado diretamente em sistemas distribuídos que se encontram de fato em produção.

PROBLEMA DA ELEIÇÃO DE LÍDER. Uma topologia de rede unidirecional em anel consiste de n processadores dispostos de tal forma que o processador P_i recebe mensagens diretas apenas do processador P_{i-1} e só envia mensagens diretamente para o processador P_{i+1} , sendo as adições e subtrações feitas módulo n . Tais redes, de maneira geral, necessitam de um processador distinto para coordená-las. Tal processador, chamado de líder da rede, é responsável por tarefas como organizar o acesso a recursos compartilhados ou prover serviços requeridos por algum processador. Caso o líder atual seja desativado, é preciso escolher algum processo para substituí-lo.

De maneira sucinta, o problema consiste em desenvolver um protocolo a ser executado por todos os processos para que algum deles seja eleito como líder caso a rede esteja desprovida de um — o líder atual foi desativado ou a rede se encontra em seu estado inicial, por exemplo.

3.2. Restrições e propriedades da solução. Como no problema anterior, estamos interessados em uma solução distribuída e simétrica. Se fôssemos considerar as soluções que não estejam completamente distribuídas, isto é, em que há um processador controlador, o problema sequer existe, já que o processador controlador pode ser discriminado dos outros e é, por definição, o líder.

Para redes em que exista alguma assimetria, especialmente alguma que forneça para cada processador um identificador único (possivelmente retirado de um conjunto ordenado), o problema de escolher um líder se resume a escolher o processador com o maior — ou menor — identificador. Para mais detalhes, veja Chang e Roberts (1979) e Peterson (1982).

Para fins de simplicidade, vamos pressupor que a rede trabalha de maneira síncrona. Em uma rede síncrona o tempo é dividido em infinitos segmentos $0, 1, 2, 3, \dots$. O processamento é iniciado no segmento 0 e em cada segmento de tempo t um processador P_i é capaz de receber uma mensagem que tenha sido enviada pelo processador P_{i-1} no segmento de tempo $t - 1$, efetuar algum processamento e então enviar uma mensagem para P_{i+1} . O problema também pode ser resolvido para redes assíncronas, veja a seção 3 de Itai e Rodeh (1990) para uma solução.

Outra suposição prévia é a de que todos os processadores conhecem o tamanho da rede. Ainda que pareça contraditório supor que os processadores sejam incapazes de se identificar e mesmo assim saibam o número de nodos da rede, consideraremos que o problema de determinar o tamanho de uma rede cuja topologia é um anel unidirecional é distinto do problema de eleger um líder. Para uma solução desse problema, veja a seção 4 de Itai e Rodeh (1990).

3.3. Não-existência de solução determinística. Em Angluin (1980), provou-se que não há solução determinística para o problema em questão. A prova é bastante semelhante àquela para a inexistência de solução determinística para o problema do Jantar dos Filósofos e parte do princípio que, sendo os processadores inicialmente idênticos, a sincronia impõe a igualdade de estados para todos os processadores. Para redes assíncronas, pode-se assumir a existência de um escalonador malicioso que disponha de poder suficiente para forçar a sincronia. Se não há um processador distinto, não há líder e portanto não há solução determinística para o problema.

3.4. Solução probabilística. Novamente precisamos recorrer aos algoritmos probabilísticos para resolver o problema, visando fugir da simetria. Para esse problema, entretanto, é impossível desenvolver uma solução que termine certamente e com uma resposta correta. Dessa maneira, sacrificaremos a certeza de término para garantir a exatidão, fazendo uso de uma solução que termine com alta probabilidade e, sempre que termine, forneça uma resposta correta.

Para fins de compreensão, consideraremos um algoritmo como um diagrama de estados dos processadores. A cada momento t o estado da rede é completamente determinado pelo estado de cada processador e pelas mensagens transmitidas no momento $t - 1$. Cada processador possui um *estado de parada*, a partir do qual todas as transições de estado levam ao próprio estado e não há mais envio de mensagens. Um algoritmo *termina* quando todos os processadores chegam ao estado de parada. A complexidade, em termos de tempo, se dá pelo número de segmentos de tempo decorridos até o término do algoritmo. Em termos de mensagens, a complexidade é o número total de mensagens enviadas por todos os processos até o término do algoritmo.

Para adicionar aleatoriedade ao protocolo, vamos supor que apesar de idênticos, cada processador dispõe de um gerador independente de números aleatórios. Representaremos este gerador por uma seqüência infinita $\{r(t) \in \mathbb{R}: 0 \leq r(t) \leq 1\}_{t=1}^{\infty}$. A complexidade de um algoritmo A , em termos de mensagens, depende então de uma função de r .

O algoritmo a seguir, denominado *Algoritmo para Eleição de Líder*, como apresentado em Itai e Rodeh (1990), nos fornece uma solução probabilística para o problema em questão.

O funcionamento se dá em fases de n segmentos de tempo. A cada fase, $a \leq n$ processadores estão *ativos*. A cada fase, alguns dos processadores ativos podem se desativar, de maneira que finalmente se chegará a um ponto em que $a = 1$ e o líder está eleito. No começo de cada fase, cada processador ativo decide com probabilidade a^{-1} se irá se tornar um candidato. Ao final de cada fase, todos os processadores sabem o total c de candidatos. Caso $c = 1$, o único candidato torna-se o líder. Se $c > 1$, os processadores ativos na fase seguinte são os candidatos da fase atual. Por fim, se $c = 0$ então a fase atual foi *inútil* e todos os processadores ativos mantêm-se ativos.

Algoritmo 2 Eleição de Líder

```
1: {Inicialização}
2: tempo := 0;
3: ativo := VERDADEIRO;
4: candidato := VERDADEIRO; { $n$  é o número total de processadores}
5:  $a := n$ ;
6:  $c := n$ ;
7: enquanto VERDADEIRO
8:   se tempo não é divisível por  $n$ 
9:     se existe mensagem na fila
10:      enviar mensagem adiante;
11:       $c := c + 1$ ;
12:   senão
13:     {se tempo é divisível por  $n$ , incluindo tempo = 0}
14:     se  $c = 1$ 
15:       se candidato
16:         terminar('Sou o líder');
17:       senão
18:         terminar('Não sou o líder');
19:     senão
20:       se  $c > 1$ 
21:          $a := c$ ;
22:         ativo := candidato
23:         {o trecho a seguir é executado para  $c = 0$  e  $c > 1$ }
24:         se ativo e  $r(\text{tempo}) \leq a^{-1}$ 
25:           candidato := VERDADEIRO;
26:            $c := 1$ ;
27:           enviar mensagem;
28:         senão
29:           candidato := FALSO
30:            $c := 0$ ;
31:       aguardar fim do segmento de tempo;
32:     tempo := tempo + 1;
```

Para o tempo 0, todos os processadores se tornam ativos, logo $a = n$. Para que se calcule c , cada candidato envia uma mensagem ao começo da fase. A mensagem é enviada através do anel, retornando ao fim da fase para seu originador. Para um processador deduzir o número de candidatos, basta contar o número de mensagens recebidas.

Vamos agora calcular a complexidade de tempo esperada para o algoritmo sugerido. Seja $p(a, c)$ a probabilidade de que c dentre a processadores ativos se tornem candidatos. Uma vez que cada um dos a processadores ativos decide se tornar candidato com probabilidade a^{-1} , temos

$$p(a, c) = \binom{a}{c} a^{-c} \left(1 - \frac{1}{a}\right)^{a-c} = \left(1 - \frac{1}{a}\right)^a \binom{a}{c} (a-1)^{-c}.$$

Seja $l(a)$ o número esperado de fases necessárias para que o número de processadores seja reduzido de a para um. Temos que

$$l(a) = 1 + p(a, 0)l(a) + \sum_{k=1}^a p(a, k)l(k)$$

e, por definição, $l(1) = 0$. Logo,

$$\begin{aligned} l(a) - l(a)p(a, 0) - l(a)p(a, a) &= 1 + \sum_{k=2}^{a-1} p(a, k)l(k) \Rightarrow \\ l(a)(1 - p(a, 0) - p(a, a)) &= 1 + \sum_{k=2}^{a-1} p(a, k)l(k) \Rightarrow \\ (1) \quad l(a) &= \frac{1 + \sum_{k=2}^{a-1} p(a, k)l(k)}{1 - p(a, 0) - p(a, a)}. \end{aligned}$$

Faremos agora uma série de observações que resultarão no teorema 18, apresentado na página 30.

LEMA 7. *Para a notação acima temos $l(a) < e$, com $e = 2,71828\dots$*

DEMONSTRAÇÃO. Indução em $a \geq 1$.

Base: $l(1) = 0$.

Passo indutivo: Pela equação (1) e pela hipótese indutiva, temos que:

$$\begin{aligned} l(a) &\leq \frac{1 + e(1 - p(a, 0) - p(a, 1) - p(a, a))}{1 - p(a, 0) - p(a, a)} \\ &= e + \frac{1 - p(a, 1)e}{1 - p(a, 0) - p(a, a)} < e. \end{aligned}$$

A última desigualdade vale uma vez que¹ $p(a, 1) = (a - \frac{1}{a})^a \frac{a}{a-1}$ é uma seqüência monotônica decrescente que converge para e^{-1} , portanto $p(a, 1)e > 1$. \square

PROPOSIÇÃO 8. *Para todo $c \leq a$ vale que $\lim_{a \rightarrow \infty} p(a, c) = (e \cdot c!)^{-1}$.*

DEMONSTRAÇÃO. A probabilidade $p(a, c)$ é um produto de dois fatores $(1 - \frac{1}{a})^a$ e $\binom{a}{c} (a-1)^{-c}$. O primeiro converge para e^{-1} , como visto anteriormente. Para um c fixo, o segundo converge para $(c!)^{-1}$. \square

Seja $d(a) = 1 - p(a, 0) - p(a, a)$ a probabilidade de que pelo menos um dos processadores se candidate e pelo menos um dos processadores não se candidate. Então o número esperado de fases para que se eleja um líder pode ser escrito como $l(a) = \frac{1 + \sum_{k=2}^{a-1} p(a, k)l(k)}{d(a)}$.

PROPOSIÇÃO 9. *A probabilidade $d(a)$ converge para $d(\infty) = 1 - e^{-1}$.*

DEMONSTRAÇÃO. Por definição, $d(a) = 1 - p(a, 0) - p(a, a)$. Sabemos que $p(a, 0) = (1 - \frac{1}{a})^a$ converge para e^{-1} e $p(a, a) = a^{-a}$ converge para zero. \square

PROPOSIÇÃO 10. *Para $a \geq 2$, vale que $d(a) \geq \frac{1}{2}$.*

DEMONSTRAÇÃO. Claramente, $d(2) = \frac{1}{2}$. Para $a \geq 3$, como $(1 - \frac{1}{a})^a$ converge para e^{-1} , temos que

$$d(a) = 1 - (1 - \frac{1}{a})^a - (\frac{1}{a})^a > 1 - e^{-1} - \frac{1}{3} > \frac{1}{2}.$$

\square

Seja $L(a, c) = \frac{1 + \sum_{k=2}^c p(a, k)l(k)}{d(a)}$ o número esperado de fases para que se eleja um líder, considerando que no máximo c processadores se candidataram na fase inicial.

¹veja apêndice 1

PROPOSIÇÃO 11. *Para c fixo, $L(a, c)$ converge como função de a .*

DEMONSTRAÇÃO. Pela proposição 8, $p(a, k)$ converge. Pela proposição 9, $d(a)$ também o faz e pela proposição 10 $d(a)$ está limitado de maneira a não alcançar 0. \square

PROPOSIÇÃO 12. *Para $3 \leq c \leq a$, vale que $p(a, c) \leq \frac{e^{-1}}{c!}$.*

DEMONSTRAÇÃO. Da definição de $p(a, c)$, da definição de coeficiente binomial e de $(1 - 1/a)^a < e^{-1}$, temos

$$p(a, c) = \left(1 - \frac{1}{a}\right)^a \binom{a}{c} (a-1)^{-c} < e^{-1} \frac{a(a-1) \cdots (a-c+1)}{(a-1)^c} \cdot \frac{1}{c!}.$$

$$\text{Portanto } p(a, c) < e^{-1} \frac{a(a-2)}{(a-1)^2} \cdot \frac{1}{c!} < \frac{e^{-1}}{c!}.$$

\square

PROPOSIÇÃO 13. *Para $c < a$, vale que $0 < l(a) - L(a, c) < \frac{4}{c!}$.*

DEMONSTRAÇÃO. Por definição,

$$l(a) - L(a, c) = \frac{\sum_{k=c+1}^{a-1} p(a, k) l(k)}{d(a)} > 0$$

A soma é positiva uma vez que todos os termos são positivos. Pelo lema 7, temos $l(k) < e$.

Pela proposição 10, temos $\frac{1}{d(a)} \leq 2$. Portanto

$$l(a) - L(a, c) < 2e \sum_{k=c+1}^{a-1} p(a, k).$$

Pela proposição 12,

$$\begin{aligned} l(a) - L(a, c) &< 2e \sum_{k=c+1}^{\infty} \frac{e^{-1}}{k!} = 2 \sum_{k=c+1}^{\infty} \frac{1}{k!} \\ &< \frac{2}{c!} \sum_{k=1}^{\infty} \frac{1}{k!} = \frac{2}{c!} (e - 1) < \frac{4}{c!}. \end{aligned}$$

\square

Seja $L(\infty, c) = \lim_{a \rightarrow \infty} L(a, c)$.

PROPOSIÇÃO 14. *Para todo c existe a_c tal que para todo $a \geq a_c$ vale que*

$$|l(a) - L(\infty, c)| < \frac{8}{c!}.$$

DEMONSTRAÇÃO. Pela proposição 11, para a suficientemente grande, temos $|L(a, c) - L(\infty, c)| < 4/c!$. Pela proposição 13, temos $|l(a) - L(a, c)| < \frac{4}{c!}$. Portanto, pela desigualdade triangular, temos

$$|l(a) - L(\infty, c)| \leq |l(a) - L(a, c)| + |L(a, c) - L(\infty, c)| < \frac{4}{c!} + \frac{4}{c!} = \frac{8}{c!}$$

□

PROPOSIÇÃO 15. *$L(\infty, c)$ é uma seqüência convergente de c .*

DEMONSTRAÇÃO. Pelo critério de convergência de Cauchy² basta mostrar que para todo $\varepsilon > 0$ existe c tal que, se $c_1, c_2 \geq c$, então $|L(\infty, c_1) - L(\infty, c_2)| < \varepsilon$. Escolha c tal que $\frac{16}{c!} < \varepsilon$. Sejam $c_1, c_2 > c$. Para a suficientemente grande, temos

$$|l(a) - L(\infty, c_1)| < \frac{8}{c_1!} \text{ e } |l(a) - L(\infty, c_2)| < \frac{8}{c_2!}$$

segundo a proposição 14. Então

$$\begin{aligned} |L(\infty, c_1) - L(\infty, c_2)| &= |L(\infty, c_1) - l(a) + l(a) - L(\infty, c_2)| \\ &\leq |L(\infty, c_1) - l(a)| + |l(a) - L(\infty, c_2)| \\ &< \frac{16}{c!} < \varepsilon, \end{aligned}$$

de maneira que provamos a proposição. □

Seja $L = \lim_{c \rightarrow \infty} L(\infty, c)$. Então, pela proposição 14, temos o seguinte resultado.

LEMA 16. *$l(a)$ converge e seu limite é L .*

PROPOSIÇÃO 17. *A esperança $L(\infty, c)$ converge para L com velocidade inversamente proporcional a $c!$, mais especificamente $|L - L(\infty, c)| < \frac{4}{c!}$.*

²veja apêndice 2

DEMONSTRAÇÃO. Por definição,

$$L(\infty, c) = \frac{1 + \sum_{k=2}^c p(\infty, k)l(k)}{d(\infty)} \text{ e } L = \frac{1 + \sum_{k=2}^{\infty} p(\infty, k)l(k)}{d(\infty)}.$$

Então

$$L - L(\infty, c) = \frac{1 + \sum_{k=c+1}^{\infty} p(\infty, k)l(k)}{d(\infty)}.$$

Portanto, $L - L(\infty, c) \geq 0$. Pela proposição 9, $d(\infty) = 1 - e^{-1} > \frac{1}{2}$. Usando o lema 7 e a proposição 8, temos

$$0 \leq |L - L(\infty, c)| < \frac{e}{\frac{1}{2}} \sum_{k=c+1}^{\infty} \frac{e^{-1}}{k!} < \frac{2(e-1)}{c!} < \frac{4}{c!}.$$

□

TEOREMA 18. *Se $l(a)$ é o número esperado de fases necessárias para reduzir o número de processadores ativos de a para um, então $l(a)$ converge para 2.441716...*

DEMONSTRAÇÃO. Pelo lema 16, temos que $l(a)$ converge para L . Pelo lema 17, temos $|L - L(\infty, 12)| < \frac{4}{12!} < 10^{-7}$. Através de cálculos elementares, obtemos $L(\infty, 12) \approx 2.441716$. □

COROLÁRIO 19. *Um anel síncrono com n processadores que saibam o valor de n escolhe um líder distribuídamente em tempo Ln , onde $L \approx 2.441716$. A complexidade média em termos de mensagens enviadas é Ln .*

DEMONSTRAÇÃO. Um processador ativo P_i manda uma mensagem apenas se $r(t) \leq a^{-1}$. Portanto, a probabilidade de que algum processador P_i envie uma mensagem é a^{-1} e o número esperado de mensagens enviadas por fase é

$$\sum_{k=0}^a kp(a, k) = a \cdot a^{-1} = 1.$$

Logo, o número esperado de mensagens enviadas por fase é n . O corolário é verdadeiro uma vez que o número esperado de fases converge para L e cada fase requer n segmentos de tempo. □

3.5. Conclusões. Mais uma vez mostramos uma situação onde a quebra de simetria é essencial para a resolução do problema. Não só é impossível desenvolver uma solução determinística, mas também mostramos que a solução probabilística possui complexidade $O(n)$, o que a torna bastante rápida.

As provas aqui apresentadas foram baseadas na seção 2.3 de Itai e Rodeh (1990). Na seção 2.4 desse artigo encontra-se outra solução ainda mais eficiente.

4. O problema de Roteamento em um n -cubo

4.1. Introdução. Nesta seção será apresentado o problema do *Roteamento em um n -cubo*. Primeiramente fornecerei uma motivação para o estudo do problema, bem como seu enunciado. Explicarei então o funcionamento da solução proposta em Valiant e Brebner (1981) e seus resultados em termos de complexidade.

Assim como o problema anterior, esse também envolve uma situação prática: a topologia do n -cubo é bastante usual em computadores paralelos. Para um computador que disponha de N processadores, modelamos a rede em que se encontram como um grafo $G = (V, E)$, onde os processadores são representados pelos vértices e as ligações entre processadores pelas arestas. Naturalmente, desejamos uma rede em que:

- (a) Os vértices possuam um grau baixo, uma vez que maior grau significa uma necessidade de mais ligações entre processadores.
- (b) A distância máxima entre dois vértices de G — seu *diâmetro* — seja pequena, para que dois processadores que não estejam diretamente ligados possam se comunicar rapidamente.
- (c) Os processadores sejam simples e idênticos, para facilitar o desenvolvimento de um algoritmo de processamento.

Um grafo que preenche esses requisitos de maneira satisfatória é o *hipercubo*, definido a seguir.

Definição. Um *hipercubo* n -dimensional é um grafo $H_n = (V, E)$, onde V é um conjunto de vértices tal que $|V| = N = 2^n$, para algum n inteiro positivo, E é o conjunto de arestas tal que cada vértice $i \in V = \{0, 1, \dots, N - 1\}$ tem exatamente $n = \log_2 N$ vizinhos e os vértices

i e j são vizinhos se e somente se suas representações binárias diferem em exatamente uma coordenada. Um hipercubo n -dimensional é usualmente chamado n -cubo.

PROBLEMA DO ROTEAMENTO EM UM n -CUBO. *Tomemos um computador paralelo cujos processadores estão dispostos em uma rede de topologia em n -cubo. A rede utiliza o modelo full-duplex, em que cada ligação entre um par de processadores pode transmitir exatamente uma mensagem através de cada um de seus sentidos. Consideramos a situação em que inicialmente cada processador possui uma mensagem a ser enviado a algum processador e que cada processador é o destino de exatamente uma mensagem. O problema consiste em desenvolver um algoritmo de roteamento a ser executado por todos os processadores para que as mensagens cheguem ao seu destino final. Em outras palavras, o algoritmo de roteamento deve especificar uma rota entre o processador de origem e o processador de destino para cada mensagem. Uma rota é uma seqüência de arestas que uma mensagem pode seguir para sair de seu processador de origem e chegar ao seu processador de destino.*

4.2. Restrições e propriedades da solução. Assim como nos problemas anteriores, desejamos soluções distribuídas e simétricas.

Denotando por $v(i)$ a mensagem originária de um processador i e por $d(i) \in \{0, 1, \dots, N-1\}$ o seu destino. Seja uma requisição de mensagens d uma n -upla ordenada composta por todos os $d(i)$, então estamos interessados no caso em que d é uma permutação de $\{0, 1, \dots, N-1\}$. Em outras palavras, cada processador é a origem e o destino de exatamente uma mensagem.

Por fim, a solução deve ser *desavisada*. Um algoritmo de roteamento desavisado satisfaz a propriedade de que a rota para uma mensagem $v(i)$ depende apenas de $d(i)$, não sendo influenciada por qualquer $d(j)$, quando $i \neq j$. Naturalmente, apesar de a rota em si não depender dos destinos de outras mensagens, o tempo necessário para atravessá-la depende. Como uma aresta só pode ser atravessada por uma mensagem em cada sentido a cada unidade de tempo, caso duas mensagens compartilhem uma aresta em suas rotas e acabem por tentar atravessá-la no mesmo sentido ao mesmo tempo, alguma das duas mensagens será atrasada.

Algoritmos desavisados são interessantes pois são simples de se implementar, já que o vértice atual em que a mensagem se encontra consegue determinar a próxima ligação a ser seguida simplesmente pela observação de seu destino.

4.3. Análise da limitação de soluções determinísticas. Borodin e Hopcroft (1985) provaram o seguinte teorema.

TEOREMA 20. *Seja $G = (V, E)$ um grafo com N vértices e grau mínimo δ . Para todo algoritmo de roteamento determinístico e desavisado em G , existe uma requisição de mensagens que é uma permutação de V e requer $\Omega(\sqrt{N/\delta^3})$ passos.*

DEMONSTRAÇÃO. Para um raciocínio inicial, vamos considerar um tipo de protocolo mais restrito, em que o próximo passo de uma mensagem depende apenas do vértice em que se encontra e do seu destino. Note que esse tipo de protocolo não é desavisado, uma vez que a rota de uma mensagem não depende apenas de seu destino. Para essa classe, o conjunto de rotas para uma mensagem dado um destino fixo forma uma árvore, uma vez que para cada vértice existe apenas uma aresta que a mensagem atravessará. Seguir a seqüência de arestas a partir de qualquer vértice deve levar a mensagem ao destino.

Considerando agora os algoritmos desavisados, o conjunto de rotas para um destino específico não necessariamente forma uma árvore. Temos agora um grafo dirigido G' , não necessariamente acíclico, que possui um vértice distinto v_0 , o destino. Os vértices do grafo G' têm grau de entrada máximo δ e N rotas distintas em termos de origem e não necessariamente disjuntas. Vamos chamar esse grafo de um grafo de destino $G'(\delta, N, v_0)$. Se u é a origem de uma rota que atravessa v , chamaremos u de *origem* para v .

Pretendemos encontrar um vértice que muitas mensagens precisem atravessar. Se t mensagens precisam atravessar o vértice em questão, então como no máximo δ mensagens podem entrar em um vértice a cada momento, um atraso de pelo menos $\frac{t}{\delta}$ seria necessário.

Se sobrepusermos os N grafos de destino $G'(\delta, N, v)$ determinados pelos N possíveis destinos $v \in V$ para mensagens, então cada vértice está em N grafos. Para forçar que uma mensagem com um dado destino passe por um vértice específico v , precisamos que o vértice

inicial desta seja uma origem para v no grafo de destino. Mas é possível que existam poucas origens para v e se muitos grafos de destino possuem o mesmo conjunto dessas origens é possível que sua quantidade não seja suficiente para que cada mensagem inicie em um vértice distinto. Portanto, gostaríamos de encontrar um vértice v com a propriedade de que v possui tantas origens quanto grafos de destino.

Para cada par de vértices i e j seja P_{ij} o caminho escolhido por $v(i)$ quando $d(i) = j$. Chamaremos um vértice u de uma j -origem para o vértice v se v se encontra em P_{uj} .

LEMA 21. *Seja j um vértice. Seja $0 \leq k \leq (N - 1)$. Temos então no mínimo $\frac{N}{\delta k}$ vértices cuja quantidade de j -origens é pelo menos k .*

DEMONSTRAÇÃO. Seja $S_k = \{v \in V \mid \text{o número de } j\text{-origens para } v \text{ é pelo menos } k\}$. Seja então $s_k = |S_k|$. É fácil ver que $j \in S_k$ para todo k , já que o número de j -origens para j é N . Logo,

$$\forall i \notin S_k, \exists v \in S_k, x \notin S_k: (x, v) \in P_{ij}.$$

Tal vértice x é chamado de $ult(i)$. É possível que $i = ult(i)$. Se $x = ult(i)$, então x é um vizinho de algum vértice de S_k . Cada vértice tem, por hipótese, δ vizinhos. Portanto o número de vértices x que podem ser $ult(i)$ para qualquer $i \notin S_k$ não pode ser maior do que δs_k .

De $x \notin S_k$ temos que a quantidade de j -origens para x é menor do que k . Se $x = ult(i)$, então i é uma j -origem para x . Como $x \notin S_k$, podem existir no máximo $(k - 1)$ vértices $i \notin S_k$ tais que $x = ult(i)$. Concluimos que o número de vértices que não pertencem a S_k é no máximo $(k - 1)\delta s_k$. Portanto

$$N - s_k \leq (k - 1)\delta s_k \Rightarrow$$

$$N \leq s_k(1 + (k - 1)\delta) = s_k(k\delta + 1 - \delta) \leq s_k k \delta \quad (\text{pois } \delta \geq 1).$$

Donde temos que $s_k \geq N/\delta k$, como queríamos demonstrar. \square

Como estamos interessados apenas em provar um limite inferior para todo k , vamos tomar um valor conveniente para k , por exemplo $k = \sqrt{N/\delta}$, que nos dá $N/\delta k = k$. Vamos chamar um vértice v de j -popular se a quantidade de j -origens para v é pelo menos k . Pelo lema 21, para cada um j dos N possíveis, existem pelo menos k vértices j -populares.

O número total de vértices j -populares, para $0 \leq j \leq (N-1)$, é pelo menos Nk . Portanto algum vértice v é j -popular para pelo menos $\lceil k \rceil$ diferentes vértices j . Vamos renomear os vértices para que v seja j -popular para $1 \leq j \leq \lceil k \rceil$.

Para $1 \leq j \leq \lceil k \rceil$, seja S_j o conjunto de j -origens para v . Temos que $|S_j| \geq k$. Podemos encontrar $\lceil k \rceil$ elementos distintos $x_1, x_2, \dots, x_{\lceil k \rceil}$ tais que $x_j \in S_j$, para todo j . Seja d uma permutação qualquer dos vértices tal que x_j é mapeado para j para todo j , $1 \leq j \leq \lceil k \rceil$. Podemos expressar isso como:

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_{\lceil k \rceil} & \cdots \\ 1 & 2 & \cdots & \lceil k \rceil & \cdots \end{pmatrix}$$

Para cada $j = 1, 2, \dots, \lceil k \rceil$, a rota de x_j para j passa por v . Como $\lceil k \rceil$ mensagens desejam entrar em v , o atraso necessário é pelo menos $\frac{k}{\delta} = \frac{\sqrt{N/\delta}}{\delta} = \sqrt{\frac{N}{\delta^3}}$, o que prova o teorema. \square

Note que o teorema vale para qualquer rede com N vértices e grau mínimo δ e não apenas para hipercubos.

Pelo teorema, temos que em um n -cubo existe uma permutação que requer $\Omega\left(\sqrt{N/n^3}\right)$ passos. A seguir, será apresentada a solução probabilística proposta em Valiant e Brebner (1981), bem como sua análise. Provaremos que a solução resolve o problema em $O(\log N)$.

4.4. Solução probabilística. Diferentemente dos problemas anteriores, em que usamos algoritmos probabilísticos para resolver um problema que sem essa ferramenta seria insolúvel, a finalidade aqui é evitar o pior caso, à semelhança do *Quicksort probabilístico*.

A solução, chamada de *Roteamento de Valiant-Brebner*, funciona em duas fases, executadas para cada mensagem $v(i)$:

Fase 1: Sortear uniformemente um destino intermediário $\sigma(i)$ do conjunto

$\{0, 1, \dots, N-1\}$ e enviar $v(i)$ para $\sigma(i)$.

Fase 2: Enviar $v(i)$ para seu destino final $d(i)$.

É importante notar que o envio da mensagem em cada fase é determinístico. A aleatoriedade existe apenas no sorteio do destino intermediário.

O algoritmo 3 é usado para enviar uma mensagem de um vértice a outro baseia-se na “correção” dos bits que representam o vértice atual e o destino. Em outras palavras, vamos

considerar o endereço de cada vértice como um vetor de n bits. O algoritmo proposto por Valiant para a definição de uma rota entre i e $\sigma(i)$ consiste em comparar bit-a-bit, da esquerda para a direita, o endereço do vértice atual (inicialmente, i) com o endereço de $\sigma(i)$. Caso os endereços sejam diferentes no l -ésimo bit, manda-se a mensagem através da ligação (aresta) que incide sobre o vizinho cujo endereço difere do endereço do vértice atual no l -ésimo bit — lembre-se da definição de n -cubo. A isso se dá o nome de *corrigir os bits da esquerda para a direita*.

Para a fase 2, os bits são corrigidos da direita para a esquerda.

Algoritmo 3 Envio de mensagem por correção de bits

- 1: \bar{a} é o vértice de origem;
 - 2: \bar{b} é o vértice de destino;
 - 3: **para** $l := 1$ até n
 - 4: **se** $a_l \neq b_l$
 - 5: atravessar a aresta $(b_1, \dots, b_{l-1}, a_l, \dots, a_n) \rightarrow (b_1, \dots, b_l, a_{l+1}, \dots, a_n)$;
-

Para exemplificar, imagine que $i = 1010110$, $\sigma(i) = 1011011$ e $d(i) = 0010110$. Assim sendo, a mensagem cruzaria as seguintes arestas:

Fase 1: $\{1010110, 1011110\}, \{1011110, 1011010\}, \{1011010, 1011011\}$.

Fase 2: $\{1011011, 1011010\}, \{1011010, 1011110\}, \{1011110, 1010110\}, \{1010110, 0010110\}$.

Note que é possível que na segunda fase a mensagem passe por vértices que já foram atravessados durante a primeira.

Além disso, como $\sigma(i)$ é sorteado aleatoriamente, é possível que $i \neq j$ mas $\sigma(i) = \sigma(j)$, de maneira que não necessariamente os destinos intermediários são uma permutação de todos os vértices.

Para o enfileiramento usaremos a estratégia *FIFO* (Fist-In-First-Out). Cada vértice mantém $n = \log_2 N$, uma para cada aresta de saída. Empates são decididos arbitrariamente.

Vamos definir o *atraso* de $v(i)$, $\text{atraso}(v(i))$, como o total de unidades de tempo em que $v(i)$ passa esperando em filas. O tempo total que $v(i)$ leva para alcançar $\sigma(i)$ é igual à

distância entre i e $\sigma(i)$ mais o atraso sofrido por $v(i)$. Para calcular o atraso sofrido por $v(i)$, primeiramente provaremos dois lemas.

LEMA 22. *Para a fase 1, suponha um par de caminhos compartilhem um arco, quando vistos como caminhos dirigidos. Então, uma vez que os caminhos se separem, não voltam a se juntar.*

DEMONSTRAÇÃO. Suponha que os caminhos compartilham o arco (u, v) mas separam-se em v . Um dos caminhos continua por w e o outro por $z \neq w$. Suponha que os endereços de u e v diferem na i -ésima posição de sua representação binária a partir da esquerda, enquanto o endereço v difere do de w na s -ésima e do de z na t -ésima posições. Como $w \neq z$, $s \neq t$. Sem que se perca a generalidade, suponha que $s < t$. Como os bits são corrigidos da esquerda para a direita na fase 1, temos também que $i < s$.

Os primeiros s bits de w diferem dos primeiros s bits de z , uma vez que o s -ésimo bit do endereço foi modificado ao se enviar a mensagem de v para w , mas manteve-se intacto para a mensagem que trafegou de v para z , uma vez que $s < t$. Futuramente, apenas bits cuja posição é maior do que s serão modificados para qualquer das mensagens. Desta maneira, as mensagens nunca chegarão a um mesmo vértice, ou seja, os caminhos não se juntam. \square

LEMA 23. *Seja p_i o caminho de i para $\sigma(i)$ na fase 1. Escolha o vértice i e defina $H_{ij} = 1$ se p_i e p_j compartilham pelo menos um arco e zero caso contrário. Então o atraso de $v(i)$ é:*

$$\text{atraso}(v(i)) \leq \sum_{j \neq i} H_{ij}.$$

DEMONSTRAÇÃO. Sejam e_1, e_2, \dots, e_k as arestas percorridas por $v(i)$ no caminho p_i , vistas como arestas dirigidas (arcos). Seja $S = \{v(j) | j \neq i, p_j \text{ e } p_i \text{ compartilham um arco}\}$.

Pelo lema 22, uma vez que um caminhos p_j , para $v(j)$ em S , se separa de p_i , eles nunca voltam a se juntar. Vamos dizer que uma mensagem Y se encontra no ponto (l, t) se, imediatamente antes do instante t , ela se encontra esperando na fila para e_l . Para qualquer inteiro h , o conjunto de pontos $\{(l, t) | l - t = h\}$ é chamado de h -ésima *diagonal*. Enquanto uma mensagem se movimenta sem que espere em uma fila, ela se mantém na diagonal, uma vez que l e t são incrementados simultaneamente. Se a mensagem é atrasada, ela muda para

uma diagonal inferior, uma vez que t aumenta, mas l não. Vamos dizer que uma mensagem *termina* em (l, t) se ela atravessa o arco e_l no instante t e o próximo arco de sua rota não se encontra em p_i ou não existe. Ela *termina em uma diagonal h* se termina em um ponto (l, t) , com $l - t = h$. Pelo lema 22, uma mensagem só pode terminar em um único ponto (l, t) .

Vamos agora mostrar que, se uma mensagem $v(i)$ sofre atraso enquanto está na diagonal h , então alguma mensagem em S termina na diagonal h . Portanto $\text{atraso}(v(i)) \leq |S|$, já que cada mensagem é atrasada no máximo uma vez em cada diagonal. Suponha que $v(i)$ é atrasada em (l, t) , com $l - t = h$. Alguma mensagem Y é responsável pelo atraso de $v(i)$. Sabemos que Y também está no ponto (l, t) , caso contrário $v(i)$ não seria atrasada nesse ponto. Logo, temos que existe pelo menos uma mensagem (no caso, Y) de S que também se encontra no ponto (l, t) , na diagonal h . Seja t' o último instante em que alguma mensagem Z em S está na diagonal h . Logo Z ocupa (l', t') , com $l' - t' = h$. Seja $f = e_{l'}$. Imediatamente antes de t' , Z está prestes a atravessar f . É fácil ver que alguma mensagem W atravessa f em t' , a própria Z ou alguma outra. Entretanto, W deve terminar em (l', t') , senão ela iria se encontrar no ponto $(l' + 1, t' + 1)$ e na diagonal h após atravessar f , o que contradiz a hipótese de que t' é o último ponto em que alguma mensagem de S se encontra na diagonal h .

Desta maneira, temos que $\text{atraso}(v(i))$ é no máximo o número total de mensagens que compartilham algum arco com $v(i)$, como queríamos demonstrar. \square

Uma vez provados esses lemas, podemos partir para a análise da complexidade do algoritmo. Analisaremos cada fase separadamente.

Análise da fase 1: Escolha um vértice i . Pelo lema 23,

$$\text{atraso}(v(i)) \leq \sum_{j \neq i} H_{ij} \leq \sum_j H_{ij}.$$

Seja P o conjunto de N possíveis rotas p_i que a mensagem $v(i)$ pode tomar na fase 1. Perceba que o número de rotas possíveis é N para a fase 1 porque uma vez que $\sigma(i)$ é escolhido, a rota é fixa. Seja $D = \text{atraso}(v(i))$. Logo,

$$\begin{aligned} \mathbb{E}[D] &= \frac{1}{N} \sum_{p \in P} \mathbb{E}[D | p_i = p] \\ &\leq \max_{p \in P} \mathbb{E}[D | p_i = p]. \end{aligned}$$

Vamos definir a variável aleatória $X = n^\circ$ de rotas p_j com $j \neq i$ que compartilham algum arco com p . Pelo lema 23, temos que

$$\mathbb{E}[D] \leq \max_{p \in P} \mathbb{E}[X|p_i = p]$$

Mas a escolha de p_j é independente de p_i se $i \neq j$. Portanto, temos

$$\begin{aligned} \mathbb{E}[D] &\leq \max_{p \in P} \mathbb{E}[X] \\ &\leq \max_{p \in P} \mathbb{E}[\text{n}^\circ \text{ de rotas } p_j \text{ que compartilham algum arco com } p] \end{aligned}$$

Escolha $p \in P$. Se p não tem arestas, o valor esperado é zero. Caso contrário, seja p um caminho dirigido com k arcos e_1, e_2, \dots, e_k . Para cada j , o número de rotas $p_j, j \neq i$, tais que p_j compartilha algum arco com p é no máximo

$$\sum_j \sum_{l=1}^k \begin{cases} 1 & \text{se } p_j \text{ contém o arco } e_l \\ 0 & \text{caso contrário.} \end{cases}$$

Pela linearidade da esperança, chegamos a

$$\begin{aligned} &\max_{p \in P} \mathbb{E}[\text{n}^\circ \text{ de rotas } p_j \text{ que compartilham um arco com } p] \\ &\leq \sum_j \sum_{l=1}^k \mathbb{P}(p_j \text{ contém o arco } e_l) \\ &= \sum_{l=1}^k \sum_j \mathbb{P}(p_j \text{ contém o arco } e_l) \\ &= \sum_{l=1}^k \mathbb{E}[\text{n}^\circ \text{ de } j \text{ tal que } p_j \text{ contém o arco } e_l]. \end{aligned}$$

Por simetria, o número esperado de j tal que p_j contém e é o mesmo independentemente de qual é o arco e . Para um dado j , o tamanho esperado de p_j é $\frac{n}{2} = \frac{1}{2} \log_2 N$. A quantidade de tais índices j é N , donde temos que a esperança da soma, em j , do tamanho de p_j é $\frac{1}{2} N \log_2 N$. Isso equivale a

$$(2) \quad \mathbb{E} \left[\sum_{e \in p} (\text{n}^\circ \text{ de } j \text{ tal que } p_j \text{ contém } e) \right].$$

Mas o número de arestas é $\frac{1}{2} N \log_2 N$. Portanto a equação (2) equivale a

$$\frac{1}{2} N \log_2 N \cdot \mathbb{E}[\text{n}^\circ \text{ de } j \text{ tal que } p_j \text{ contém } e] = \frac{1}{2} N \log_2 N.$$

Desse resultado, temos que para todo e

$$\mathbb{E}[\text{n}^\circ \text{ de } j \text{ tal que } p_j \text{ contém } e] = 1.$$

Finalmente, podemos concluir que

$$\mathbb{E}[\text{n}^\circ \text{ de } p_j \text{ que compartilham um arco com } p] \leq k \leq n = \log_2 N.$$

Ou seja, o atraso esperado é no máximo $\log_2 N$. Isso completa a análise da fase 1.

A partir de agora, tratemos de $\sum_j H_{ij}$, ao invés do atraso. Temos que $\sum_j H_{ij}$ é o somatório de N variáveis aleatórias de Bernoulli independentes (sendo o valor de uma delas sempre 1). Sua esperança é inferior a $\log_2 N$. Vamos aplicar os limites de Chernoff³ a essa soma.

Seja $m = \mathbb{E}[\sum_j H_{ij}] \leq \log_2 N$. Temos então

$$\mathbb{P}\left(\sum_j H_{ij} > (1 + \delta)m\right) < \left[\frac{e^\delta}{(1 + \delta)^{(1 + \delta)}}\right]^{\log_2 N}.$$

Perceba também que $\mathbb{E}[\sum_j H_{ij}]$ pode ser menor do que $\log_2 N$. Se $(1 + \delta) \geq e$, então

$$\frac{e^\delta}{(1 + \delta)^{(1 + \delta)}} = \frac{e^\delta}{(1 + \delta)(1 + \delta)^\delta} \leq \frac{1}{1 + \delta}.$$

A análise acima independe da escolha de i . Portanto, concluimos que se $(1 + \delta) \geq e$,

$$(3) \quad \mathbb{P}(\exists v(i) : \text{atraso}(v(i)) > (1 + \delta) \log_2 N) < N \left(\frac{1}{1 + \delta}\right)^{\log_2 N}$$

Por exemplo, tomando $\delta = 7$, temos

$$\left(\frac{1}{1 + \delta}\right)^{\log_2 N} = \frac{1}{8^{\log_2 N}} = \frac{1}{N^3}.$$

Portanto, a probabilidade de que todas as mensagens cheguem em tempo menor do que

$$\underbrace{\log_2 N}_{\text{distância}} + \underbrace{8 \log_2 N}_{\text{atraso}}$$

é pelo menos $1 - \frac{1}{N^2}$.

³veja apêndice 3

Análise da fase 2: Para simplificar a análise, vamos considerar que cada mensagem $v(i)$ espera em seu destino intermediário $\sigma(i)$ até $\lceil (2 + \delta) \log_2 N \rceil$ instantes tenham se passado desde o início da execução. Só então a fase 2 começa.

A segunda fase começa com $v(i)$ em $\sigma(i)$ para todo i . Nessa fase, as mensagens se movem de $\sigma(i)$ para $d(i)$ “corrigindo” os bits da *esquerda para a direita*. Novamente, temos que o atraso de uma mensagem $v(i)$ é no máximo o número de $j \neq i$ tal que p_j e p_i compartilham um arco.

A análise dessa fase é bastante semelhante à da fase 1.

Escolha um i . Seja D' o atraso sofrido por $v(i)$ durante o seu passeio de $\sigma(i)$ até $d(i)$. Seja p'_i a rota de $\sigma(i)$ para $d(i)$. Então

$$D' \leq \sum_{j \neq i} \begin{cases} 1 & \text{se } p'_i \text{ e } p'_j \text{ compartilham algum arco} \\ 0 & \text{caso contrário.} \end{cases}$$

O caminho de $\sigma(i)$ para $d(i)$ compartilha um arco com o caminho $\sigma(j)$ para $d(j)$ se e somente se os caminhos de $d(i)$ para $\sigma(i)$ (com “correção” de bits da esquerda para a direita) e de $\sigma(j)$ para $d(j)$ (com “correção” de bits da esquerda para a direita) compartilham um arco. Seja

$$H'_{ij} = \begin{cases} 1 & \text{se } p'_i \text{ e } p'_j \text{ compartilham algum arco} \\ 0 & \text{caso contrário.} \end{cases}$$

Como na fase 1, temos que H'_{ij} são independentes para $j \neq i$. Assim sendo, a análise pode ser feita de maneira análoga à da primeira fase. Podemos então concluir que a probabilidade de que alguma mensagem não complete a segunda fase em menos de $\lceil (2 + \delta) \log_2 N \rceil$ é no máximo $N \left(\frac{1}{1+\delta}\right)^{\log_2 N}$ se $(1 + \delta) \geq e$.

Ainda que as fases 1 e 2 não sejam independentes, podemos somar as probabilidades segundo a equação (3) para concluir que $2N \left(\frac{1}{1+\delta}\right)^{\log_2 N}$ é um limite superior para a probabilidade de que alguma mensagem $v(i)$ não consiga chegar a $d(i)$ em tempo $2(2 + \delta) \log_2 N$.

Chegamos então ao seguinte resultado.

TEOREMA 24. *Para o algoritmo de roteamento de Valiant-Brebner, o tempo esperado para que as mensagens $v(i)$, para todo $i \in V$, cheguem ao seu destino $d(i)$ é $O(\log N)$.*

4.5. Conclusões. Vimos nessa seção um exemplo de como o uso de algoritmos probabilísticos pode ser usado não apenas para resolver problemas que não possuem solução determinística, mas também para o desenvolvimento de soluções mais eficientes do que as determinísticas. Os resultados são bastante expressivos, uma vez que $\Omega(\sqrt{N/n^3})$ é um limitante inferior que cresce assintoticamente muito mais rápido do que $O(\log N)$.

A prova para a limitação inferior da complexidade no caso determinístico é de autoria de Paul Beame.

5. O problema dos Generais Bizantinos

5.1. Introdução. Nessa seção veremos o problema dos *Generais Bizantinos*. Apresentarei inicialmente o enunciado do problema, para depois explicar sua representação computacional. Veremos então a solução probabilística proposta em Ben-Or (1983) e as suas limitações.

O problema dos Generais Bizantinos é fundamental para a computação distribuída, uma vez que trabalha com o consenso por parte de um conjunto de processos. Se não fôssemos capazes de garantir algum tipo de acordo entre um conjunto de processos, seria impossível coordená-los para resolver algum problema que dependa do conjunto dos resultados de suas computações. Outro motivo para que se estude o problema em questão é a sua surpreendente complexidade, quando comparada com a simplicidade de seu enunciado. Como o problema do Jantar dos Filósofos, seu enunciado é a representação antropomórfica de um problema computacional.

PROBLEMA DOS GENERAIS BIZANTINOS. n generais bizantinos precisam coordenar sua estratégia militar. Pode-se optar por atacar ou fugir das forças do inimigo que se aproxima. Cada general possui sua própria opinião a respeito de que decisão deve ser tomada. Uma vez que os exércitos se encontram distantes uns dos outros, é preciso decidir a estratégia através do envio de mensagens entre generais. Entretanto, alguns dos generais são traidores, de maneira que não é possível confiar em suas mensagens. Pode-se assumir, sem perda de generalidade, que os mensageiros são todos leais, uma vez que um general leal com um mensageiro traidor pode ser considerado um general traidor.

Pode-se então entender o problema como um problema de protocolo de consenso entre n processos completamente conectados, em que alguns dos processos podem estar operando incorretamente, talvez até trabalhando contra o sistema. Um problema de protocolo de consenso consiste de um conjunto de processos $\{P_1, P_2, \dots, P_n\}$. Cada processo P_i possui uma variável booleana x_i , cujo valor inicial é b_i . Deve-se então desenvolver um protocolo a ser executado por todos os processos para que as seguintes condições sejam satisfeitas ao seu término:

Acordo Todos os processos decidem um mesmo valor v . Ao “decidir v ”, um processo grava v em um registrador privado de escrita única. Desta maneira, temos que após sua decisão um processo não pode mais mudá-la.

Validade Se todos os processos corretos começam com o valor v em x_i , a sua decisão final deve ser v .

A condição de validade nos impede de recorrer à solução trivial, isto é, aquela em que todos os processos decidem um valor pré-determinado.

5.2. Restrições e propriedades da solução. Assim como nos três problemas anteriores, desejamos soluções distribuídas e simétricas. A única ressalva é que os processos não são necessariamente simétricos no que diz respeito ao seu valor booleano inicial. Podemos considerar que este valor é o resultado de alguma computação anterior.

Além disso, vamos pressupor que os processos trabalham de maneira assíncrona. A nossa intenção é mostrar um caso em que não há solução sem o uso de algoritmos probabilísticos. Em Pease et al. (1980) temos uma solução determinística para o caso de processos síncronos.

Imaginamos aqui um escalonador semelhante ao definido para o problema do Jantar dos Filósofos. Considerando que os processos estão sendo executados por um processador único, precisamos definir a ordem em que os processos serão ativados. Um *escalonador* é a função responsável por essa tarefa. Daremos bastante poder ao escalonador, que será capaz de investigar o estado interno de quaisquer processos para decidir qual será ativado a seguir. Além disso, o escalonador poderá atrasar a entrega de mensagens. Ainda assim, vamos limitá-lo para que seja incapaz de interferir na leitura de bits aleatórios ou lê-los antes que algum

processo o faça. Vamos também considerar que o escalonador é *justo*, ou seja, nenhum processo fica eternamente inativo.

Também desejamos que os processos sejam completamente conectados, isto é, os processos P_i e P_j estão ligados para todos i, j , com $i \neq j$. O caso em que os processos não estão completamente conectados foi estudado em Lamport et al. (1982) e restrições sobre a conectividade necessária para a solução do problema foram impostas.

Consideramos que o meio de comunicação é tal que, se uma mensagem é enviada a partir de um processo correto para outro processo correto, ela chegará ao seu destino inalterada. Como se verá adiante, falhas no meio de comunicação podem ser encaradas como se o processo remetente fosse defeituoso.

Por fim, vamos supor que os processos defeituosos se comportam de maneira completamente imprevisível, podendo mandar mensagens segundo algum plano malicioso para prejudicar o funcionamento do sistema, anunciando decisões diferentes para processos corretos, por exemplo. Podem também não mandar mensagem alguma. Tais processos não podem interferir no comportamento de processos corretos, entretanto. Em outras palavras, um processo defeituoso não pode alterar ou apagar mensagens de processos corretos, não pode mandar mensagens fingindo ser um processo correto e nem pode alterar o código executado em um processo correto, bem como influenciar seu acesso a bits aleatórios.

5.3. Não-existência de solução determinística. Em Fischer et al. (1985) encontra-se a prova de que não há solução determinística para o problema do protocolo de consenso em sua versão assíncrona. Infelizmente a prova é muito extensa para ser completamente explicada aqui. A idéia geral por trás da prova é que existem configurações de estados para os processos a partir das quais não é possível decidir algum valor. Não apenas isso, mas temos também que é a partir de qualquer uma dessas configurações é possível chegar em outra em que a indecisão se mantém. Dessa maneira, é possível definir uma seqüência de ativação para os processos de tal modo que o sistema nunca chegará a uma configuração de estados decisiva. Seus resultados mostram que o sistema pode não chegar a um estado terminal mesmo que apenas um dos processos esteja defeituoso.

5.4. Solução probabilística. Uma solução para o problema foi proposta em Ben-Or (1983). Assim como as soluções vistas para o problema do Jantar dos Filósofos e para o problema da Eleição de Líder, a solução para o problema dos Generais Bizantinos faz uso de bits aleatórios em seu processamento. O algoritmo se aproveita do fato de que se os processos fazem escolhas aleatórias a respeito do valor que irão propor para o acordo de maneira independente, em algum momento um número suficiente de processos corretos escolherá um mesmo valor e a decisão será possível.

Cada processo correto executa o algoritmo a seguir. O algoritmo considera que no máximo t dos n processos são defeituosos. O algoritmo funciona para os casos em que $n > 5t$, como veremos mais adiante.

A variável x_i , cujo valor inicial é b_i , armazena a escolha atual do processo para valor de consenso. O algoritmo funciona em rodadas, cujo índice está armazenado em r . Cada rodada é constituída por três fases.

Na *fase de notificação*, o processo P_i anuncia o valor de x_i para todos os outros processos e então espera por $(n - t)$ mensagens de notificação. Todas as mensagens enviadas na fase de notificação carregam o valor de enumeração N .

Na *fase de proposta*, cada P_i propõe um valor de consenso retirado do conjunto $\{0, 1, ?\}$ a partir das mensagens recebidas na fase de notificação. Após enviar seu valor de proposta para todos os outros procesos, cada processo então espera por $(n - t)$ mensagens de proposta, que estão marcadas com o valor de enumeração P .

Daí, P_i propõe 0 se mais do que $\frac{(n+t)}{2}$ das mensagens de notificação recebidas contém 0. Igualmente, P_i propõe 1 se mais do que $\frac{(n+t)}{2}$ mensagens de notificação contendo o valor 1 foram recebidas. Caso nenhuma das condições acima tenham sido satisfeitas, P_i propõe “?”. Com isso, P_i está propondo que o valor de consenso seja sorteado por cada processo independentemente.

É importante perceber que P_i termina após ter enviado suas mensagens na fase de proposta caso tenha feito sua decisão na rodada anterior.

Algoritmo 4 Acordo Bizantino

```

1:  $r := 1$ ;
2: decidido := FALSO;
3: enquanto VERDADEIRO
4:   {Fase de Notificação}
5:   enviar  $(N, r, x)$  para todos os processos;
6:   esperar por  $(n - t)$  mensagens de forma  $(N, r, *)$ ;
7:   {Fase de Proposta}
8:   se mais do que  $\frac{(n+t)}{2}$  mensagens são de forma  $(N, r, w)$ , para  $w = 1$  ou  $w = 0$ 
9:     enviar  $(P, r, w)$  para todos os processos;
10:  senão
11:    enviar  $(P, r, ?)$  para todos os processos;
12:  se decidido
13:    parar;
14:  senão
15:    esperar por  $(n - t)$  mensagens de forma  $(P, r, *)$ ;
16:  {Fase de Decisão}
17:  se mais do que  $t$  mensagens são de forma  $(P, r, w)$ , para  $w = 1$  ou  $w = 0$ 
18:     $x_i := w$ ;
19:    se mais do que  $3t$  mensagens são de forma  $(P, r, w)$ 
20:      decidir  $x_i$ ;
21:      decidido := VERDADEIRO;
22:  senão
23:    sortear uniformemente um valor para  $x_i$  a partir do conjunto  $\{0, 1\}$ 
24:   $r := r + 1$ 

```

Por fim, chegamos à *fase de decisão*, em que P_i analisa as proposta recebidas para decidir um novo valor para x_i , que será usado na próxima rodada. Dependendo das propostas recebidas, é possível que P_i escreva o valor de x_i em um registrador de escrita única — P_i decide x_i .

O número da rodada r é encaminhado dentro de todas as mensagens, de maneira que os processos conseguem distinguir mensagens de rodadas diferentes. Cada um dos processos descarta mensagens de rodadas anteriores, usa as mensagens cuja rodada é a sua corrente e guarda mensagens de rodadas posteriores para que sejam usadas uma vez que a rodada apropriada seja atingida. Caso mais de uma mensagem seja recebida de um mesmo remetente para uma mesma fase de uma mesma rodada, apenas a primeira mensagem recebida é considerada, sendo as demais descartadas pelo processo. Essa política é assumida para que mensagens excedentes de processo defeituosos não atrapalhem a contagem por parte dos processos corretos. Como para qualquer rodada os processos defeituosos podem enviar mensagens com o número de rodada incorreto ou mesmo não enviar mensagem alguma, os processos corretos não devem esperar por mais do que $n - t$ mensagens em cada fase, uma vez que $n - t$ é o número mínimo de mensagens cujo recebimento é garantido.

Os lemas e o teorema a seguir, de autoria de Hadzilacos (1986), fornecem um entendimento mais profundo a respeito do funcionamento do algoritmo e provam sua efetividade.

LEMA 25. *Se um processo correto propõe um valor v em uma rodada r , então nenhum outro processo irá propor o valor $\bar{v} = 1 - v$, o complemento de v , na mesma rodada.*

DEMONSTRAÇÃO. Um processo envia mensagens (P, r, v) em sua fase de proposta caso mais do que $\frac{(n+t)}{2}$ processos tenham v como seu valor de notificação. No máximo t desses processos são defeituosos, o que nos garante que mais do que $\frac{(n+t)}{2} - t = \frac{(n-t)}{2}$ processos corretos escolheram v , ou seja, a maioria dos processos corretos escolheu v . Para que outro processo correto proponha \bar{v} na mesma rodada, a maioria dos processos corretos deveria ter escolhido \bar{v} . Como processos corretos mandam a mesma mensagem para todos os processos, tal situação é impossível. \square

LEMA 26. *Se no início de uma rodada r todos os processos P_i têm o mesmo valor v em x_i , então todos os processos corretos decidirão v em r .*

DEMONSTRAÇÃO. No começo de uma rodada, cada processo correto envia mensagens notificando os outros processos que escolheu o valor v para x_i . Cada processo correto recebe

$n - t$ mensagens, das quais no máximo t são provenientes de processos defeituosos. Temos então que cada processo recebe pelo menos $n - 2t$ mensagens de notificação provenientes de processos corretos. Como $n > 5t$ implica $n - 2t > \frac{n+t}{2}$, cada processo correto irá propor v em sua fase de proposta.

Considerando a fase de decisão, no pior caso um processo recebe t mensagens propondo \bar{v} a partir de processos defeituosos e $n - 2t$ propostas para v a partir dos processos corretos. Como $(n - 2t) > 3t$ se $n > 5t$, cada processo correto decidirá v . \square

LEMA 27. *Se um processo correto decide v na rodada r , então todos os processos corretos terão decidido v na rodada $r + 1$.*

DEMONSTRAÇÃO. Para provar este lema, basta provar que se um processo correto decide v na rodada r , então na rodada $r + 1$ todos os processos corretos irão notificar v e o lema 27 será uma consequência direta do lema 26.

Para que um processo correto P_i decida v em uma rodada r , ele precisa ter recebido mais do que $3t$ propostas para v . Como no máximo t dessas mensagens vieram de processos defeituosos, então P_i recebeu m propostas para v a partir de processos corretos, com $m > 2t$. Vamos agora analisar algum outro processo correto P_j .

O processo P_j deve ter recebido, na rodada r , propostas de $n - t$ processos. Em outras palavras, P_j recebe propostas de todos os processos, exceto t deles. Logo, dos m processos corretos dos quais P_i recebeu propostas, todos menos t deles devem ter tido suas propostas recebidas por P_j no pior caso. Porém, como $m > 2t$ implica que $m - t > t$, temos que P_j irá notificar v na próxima rodada. Daí temos que todos os processos corretos notificarão v na rodada $r + 1$. Pelo lema 26 temos que todos os processos decidirão v na rodada $r + 1$. \square

Conseguimos então o seguinte resultado a respeito do algoritmo de Ben-Or para o acordo dos generais bizantinos:

TEOREMA 28. *Assumindo que $n > 5t$, o algoritmo de Ben-Or termina com probabilidade 1 garantindo acordo e validade.*

DEMONSTRAÇÃO. A condição de acordo é satisfeita pelo lema 27 e a de validade pelo lema 26. Para que o protocolo termine, precisamos apenas que um processo decida um valor v qualquer, pois pelo lema 26 todos os processos decidirão v na rodada seguinte. Com probabilidade 1, um número suficiente de processos corretos irá sortear um mesmo valor v , se considerarmos que a computação é infinita. \square

5.5. Conclusões. O algoritmo é bastante ineficiente, uma vez que seu tempo esperado é $O(2^n)$ rodadas, que é o número esperado de arremessos de n moedas até que todas tenham o mesmo valor. Ainda assim, se o número de processadores defeituosos é $O(\sqrt{n})$, o tempo esperado de rodadas é constante. Temos aí outra vantagem para o uso de algoritmos probabilísticos, já que qualquer solução para o problema dos Gerais Bizantinos demanda pelo menos $t + 1$ rodadas (veja Fischer e Lynch (1982)), onde t é o número máximo de processos defeituosos.

Para a complexidade em termos de mensagens, temos que cada processo correto envia uma mensagem para cada um dos processos em cada rodada. Supondo que cada processo defeituoso não mande mais do que $O(n)$ mensagens por rodada, o número esperado de mensagens enviadas por rodada é $O(n^2)$.

O algoritmo de Ben-Or, juntamente com o algoritmo proposto em Rabin (1983), foi um dos primeiros a solucionar o caso assíncrono para o problema dos gerais bizantinos e é o mais simples até hoje. Para algoritmos mais elaborados, tanto em termos de eficiência quanto de tolerância ao número de processos defeituosos, veja Bracha (1985), Chor e Coan (1985) e Perry (1985).

As provas presentes nessa seção foram baseadas em Gupta et al. (1994).

Referências Bibliográficas

- Angluin, D. (1980). Local and global properties in networks of processors (extended abstract). In *STOC*, pages 82–93. ACM.
- Ben-Or, M. (1983). Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30.
- Borodin, A. e Hopcroft, J. E. (1985). Routing, merging, and sorting on parallel models of computation. *J. Comput. Syst. Sci.*, 30(1):130–145.
- Bracha, G. (1985). An $O(\lg n)$ expected rounds randomized byzantine generals protocol. In *STOC*, pages 316–326. ACM.
- Chang, E. J. H. e Roberts, R. (1979). An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM*, 22(5):281–283.
- Chor, B. e Coan, B. A. (1985). A simple and efficient randomized byzantine agreement algorithm. *IEEE Trans. Software Eng.*, 11(6):531–539.
- Dijkstra, E. W. (1971). Hierarchical ordering of sequential processes. *Acta Inf.*, 1:115–138.
- Fischer, M. J. e Lynch, N. A. (1982). A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186.
- Fischer, M. J., Lynch, N. A., e Paterson, M. (1985). Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382.
- Gupta, R., Smolka, S. A., e Bhaskar, S. (1994). On randomization in sequential and distributed algorithms. *ACM Comput. Surv.*, 26(1):7–86.
- Hadzilacos, V. (1986). Ben-or’s randomized protocol for consensus in asynchronous systems. *Course notes: Computer Science*.
- Itai, A. e Rodeh, M. (1990). Symmetry breaking in distributed networks. *Inf. Comput.*, 88(1):60–87.

- Lamport, L. e Lynch, N. A. (1990). Distributed computing: Models and methods. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 1157–1199.
- Lamport, L., Shostak, R. E., e Pease, M. C. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401.
- Lehmann, D. J. e Rabin, M. O. (1981). On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In *POPL*, pages 133–138.
- Pease, M. C., Shostak, R. E., e Lamport, L. (1980). Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234.
- Perry, K. J. (1985). Randomized byzantine agreement. *IEEE Trans. Software Eng.*, 11(6):539–546.
- Peterson, G. L. (1982). An $O(n \log n)$ unidirectional algorithm for the circular extrema problem. *ACM Trans. Program. Lang. Syst.*, 4(4):758–762.
- Rabin, M. O. (1983). Randomized byzantine generals. In *FOCS*, pages 403–409. IEEE.
- Valiant, L. G. e Brebner, G. J. (1981). Universal schemes for parallel communication. In *STOC*, pages 263–277. ACM.

Índice Remissivo

- ação atômica, 15
- Algoritmo
 - dos Filósofos Corteses, 21
 - dos Filósofos Livres, 15
 - para Eleição de Líder, 24
 - para os Generais Bizantinos, 45
- Algoritmos
 - Distribuídos, 7
 - Las Vegas, 10
 - Monte Carlo, 10
 - Probabilísticos, 9
- computação, 16
 - finita, 16
 - justa, 16
- deadlock, 17
- desavisada, 32
- disjunção
 - entre configurações de sorteios aleatórios, 19
- escalonador, 15, 43
 - justo, 16, 44
- fase
 - de decisão, 46
 - de notificação, 45
 - de proposta, 45
- hipercubo, 31
- inanição, 17
- líder de uma rede em anel, 22
- livre
 - de “deadlock”, 13
 - de travamento, 13
- Problema
 - da Eleição de Líder, 22
 - do Jantar dos Filósofos, 11
 - do Roteamento em um n -cubo, 32
 - dos Generais Bizantinos, 42
- Roteamento de Valiant-Brebner, 35
- teste de convergência de Cauchy, 55
- trancado, 17
- unidade atômica de tempo, 14

APÊNDICE A

Equações e teoremas

1. Equação para e^x

Para todo $x \in \mathbb{R}$ segue da definição de e que $e^x = \lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n$.

2. Convergência de Cauchy

O *teste de convergência de Cauchy* é um método utilizado para testar a convergência de séries infinitas. Uma série

$$\sum_{i=0}^{\infty} a_i$$

converge se e somente se para toda constante $\varepsilon > 0$ existe um número $N \in \mathbb{N}$ tal que

$$|a_m - a_n| < \varepsilon$$

vale se $m, n \geq N$.

Isso é necessário e suficiente.

3. Limite de Chernoff

Sejam X_1, X_2, \dots, X_n variáveis aleatórias de Bernoulli independentes, com

$$\mathbb{P}(X_i = 1) = p_i \text{ e } \mathbb{P}(X_i = 0) = q_i = 1 - p_i.$$

Se $X = \sum_{i=1}^n X_i$, então para qualquer $\delta > 0$ temos que

$$\mathbb{P}(X > (1 + \delta)\mathbb{E}X) < \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right)^{\mathbb{E}X}.$$