DCC207 – Algoritmos 2

Aula 14 – Soluções aproximadas para problemas difíceis (Parte 2)

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- Nessa aula, veremos outros exemplos de algoritmos aproximativos para problemas difíceis
- Veremos soluções para problemas de evidente apelo prático
- Veremos como esses problemas se relacionam com outros problemas NP-completos
- No caso específico do problema dos k-centros (clustering), veremos como a solução que iremos estudar é ótima no sentido de que nenhum outro algoritmo aproximativo possui fator de aproximação menor

Balanceamento de carga

- Considere o problema de balanceamento de carga de trabalho em múltiplos servidores homogêneos
- Nesse problema, os múltiplos servidores devem atender a uma demanda de processos a serem executados
- O objetivo nesse caso é balancear a carga de tarefas que cada servidor executará de forma a otimizar a latência ou throughput dos serviços
- Formalmente, dados m servidores M1, M2, ..., Mm, n processos a serem executados, em que cada processo j possui tempo de execução t(j), deseja-se distribuir a carga de trabalho entre as máquinas de forma mais balanceada possível

Balanceamento de carga

- Seja A(i) o conjunto de processos alocados para a máquina M_i
- A carga de trabalho de M_i é definida por $T_i = \sum t_j$, para $t_j \in A(i)$
- O objetivo do problema é minimizar a carga máxima de uma máquina, chamada de *makespan*
  - $T = \max T_i$
- Esse problema é NP-difícil, embora não demonstraremos essa afirmação
- Nosso objetivo é desenhar um algoritmo aproximativo para o balanceamento de carga dos servidores

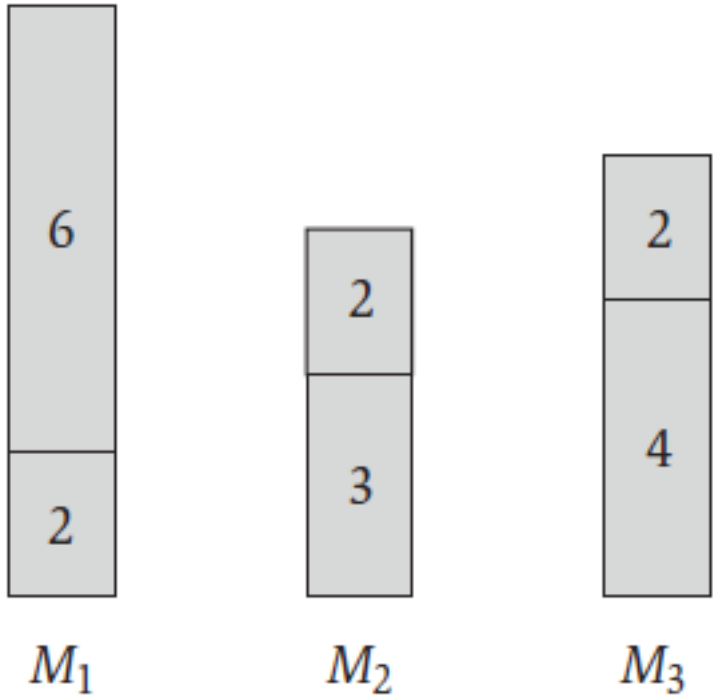
Balanceamento de carga

- Vamos analisar uma abordagem extremamente simples para o problema
- Vamos imaginar que os processos chegam em sequência, e alocamos um novo processo à máquina com a menor carga no momento

```
Greedy-Balance:
Start with no jobs assigned
Set T_i = 0 and A(i) = ∅ for all machines M_i
For j = 1, ..., n
  Let M_i be a machine that achieves the minimum min_k T_k
  Assign job j to machine M_i
  Set A(i) ← A(i) ∪ {j}
  Set T_i ← T_i + t_j
EndFor
```

Balanceamento de carga

- Exemplo:
  - 3 máquinas
  - t_j : 2, 3, 4, 6, 2, 2
  - Makespan = 8 (M1)
- O que aconteceria se os processos tivessem chegado em outra ordem?
  - Ex.: 6, 4, 3, 2, 2, 2
  - Makespan = 7
- Qual o fator de aproximação para esse algoritmo?



Balanceamento de carga

- Num cenário ideal, cada máquina ficaria responsável por 1/m da carga total dos processos
- Portanto, um limite inferior para o makespan ótimo é $1/m * \Sigma t_j$
  - $T^* \geq 1/m * \Sigma t_j$
- Agora considere uma situação em que um dos processos possui um tempo de execução relativamente maior que a soma dos tempos dos outros processos
- Se esse fosse o primeiro processo, o algoritmo guloso atingiria exatamente a solução ótima
  - Esse processo seria alocado para executar em uma máquina, e todos os demais seriam distribuídos entre as máquinas restantes
- Nesse caso particular, a estimativa do ótimo não seria muito informativa, e falharia em demonstrar o potencial do algoritmo aproximativo
- Em outras palavras, sabemos também que o makespan ótimo será, pelo menos, a carga imposta pelo processo mais demorado
  - $T^* \geq \max t_j$

Balanceamento de carga

- Teorema: O algoritmo guloso proposto para o balanceamento de carga possui fator de aproximação 2
- Prova (ideia):
  - A ideia é analisar a máquina M_i com a maior carga (a que resulta no makespan reportado)
  - Seja j o último processo alocado a M_i
  - No momento da alocação de j, Mi possuía carga T_i-t_j. Como ela era a máquina com a menor carga, sabemos que todas as demais possuíam carga maior ou igual a $T_i - t_j$
  - Logo, a soma total das cargas $\sum T_k \geq m(T_i-t_j)$; ou $T_i-t_j \leq 1/m * \sum T_k$
  - Como $\sum T_k$ é a soma total das cargas, segue que $T_i-t_j \leq T^*$
  - Como $t_j \leq T^*$, segue que $T = T_i = (T_i-t_j) + t_j \leq 2T^*$

Balanceamento de carga

- É fácil construir um exemplo em que a saída do algoritmo é próxima do fator de aproximação
- Considere que temos um total de m máquinas, e n=m(m-1)+1 processos.
- Considere ainda que os n-1 primeiros processos possuem tempo de execução 1 e o último possua tempo m
- Nesse caso, o algoritmo aloca cada um dos n-1 primeiros processos sequencialmente entre as máquinas, e uma delas termina com uma carga 2m-1, devido à alocação do último processo
- A alocação ótima seria deixar uma máquina reservada para o último processo, e distribuir igualmente a carga dos n-1 primeiros entre as m-1 máquinas restantes, resultando num makespan de m

Balanceamento de carga

- Podemos tentar melhorar o algoritmo, atacando o problema detectado no exemplo do slide anterior
- Ele ocorre porque não previámos um processo com um impacto tão grande na carga de trabalho das máquinas
- Se recebêssemos os processos em ordem decrescente de tempo de execução, poderíamos melhorar a distribuição, já que os últimos não causariam tanto impacto no tempo total
- Como veremos, essa abordagem resulta, de fato, em um fator de aproximação menor que o do algoritmo anterior

Balanceamento de carga

- Agora, vamos supor que todos os processos são disparados de uma só vez, o que nos permite escolher os mais ’pesados’ para serem executados primeiro

```
Sorted-Balance:
Start with no jobs assigned
Set T_i = 0 and A(i) = ∅ for all machines M_i
Sort jobs in decreasing order of processing times t_j
Assume that t_1 ≥ t_2 ≥ ... ≥ t_n
For j = 1, ..., n
  Let M_i be the machine that achieves the minimum min_k T_k
  Assign job j to machine M_i
  Set A(i) ← A(i) ∪ {j}
  Set T_i ← T_i + t_j
EndFor
```

Balanceamento de carga

- Vamos analisar mais um limite inferior para essa versão aprimorada
- Se tivéssemos menos processos que máquinas, trivialmente o algoritmo encontra o ótimo
- Agora vamos considerar n>m.
- Considere os m+1 primeiros processos. Todos eles possuem tempo de execução pelo menos igual a t_{m+1}.
- Como temos apenas m máquinas, dois desses processos serão alocados na mesma máquina. Consequentemente, $T^* \geq 2t_{m+1}$

Balanceamento de carga

- Teorema: O algoritmo aproximativo com ordenação dos processos possui fator de aproximação de 1.5
- Prova (ideia):
  - Novamente vamos analisar a alocação da máquina Mi com a maior carga
  - Se Mi possui apenas um processo, a alocação é ótima. Então suponhamos que Mi tenha ao menos dois processos
  - Seja j o último processo alocado em Mi. Como esse é, pelo menos, o segundo processo alocado a Mi, sabemos que $j \geq m+1$
  - Logo, $t_j \leq t_{m+1} \leq \frac{1}{2}T^*$
  - Como na análise anterior, sabemos que $T_i - t_j \leq T^*$
  - Além disso, $T = T_i = (T_i - t_j) + t_j \leq T^* + \frac{1}{2}T^* = 1.5T^*$

O problema dos k-centros (clustering)

- O problema de agrupamento é uma tarefa bastante comum em aprendizado de máquina e mineração de dados
- Essa tarefa possui diversas aplicações como:
  - Segmentação de clientes;
  - Determinar grupos de páginas web similares;
  - Detecção de grupos de pacientes com sintomas parecidos
- Existem diversas formulações para o problema de clustering
- Aqui vamos considerar uma em que é dado um conjunto de n pontos e um parâmetro k, e deseja-se encontrar k centros de forma que os pontos estejam o mais próximo possível desses centros (cada ponto está localizado a uma distância máxima r de um dos centros)

O problema dos k-centros (clustering)

- Formalmente, consideraremos a seguinte formulação:
  - Entrada: $S = \{s_1, s_2, ..., s_n\}$ um conjunto de pontos; dist: S x S -> R^+ uma função de distância (uma métrica); e um inteiro k
  - Saída: Um conjunto $C = \{c_1, c_2, ..., c_k\}$ de pontos (centros) que particiona o conjunto de pontos em k grupos (s_i pertence ao centro mais próximo)
  - Objetivo: minimizar o raio máximo dos clusters, $r(C) = \max dist(s_i, C)$
    - $dist(s_i, C) = \min dist(s_i, c_j)$

O problema dos k-centros (clustering)

- Vamos considerar uma abordagem gulosa ingênua para solucionar o problema
- Nessa abordagem, podemos escolher o posicionamento de um centro desconsiderando a colocação dos outros remanescentes
- Dessa forma, colocaríamos o primeiro centro na melhor posição considerando k=1
- Depois, adicionamos os demais centros, um a um, de forma a reduzir o raio máximo dos clusters

O problema dos k-centros (clustering)

- Essa abordagem, contudo, é muito simplória, podendo retornar resultados bastante ruins
- Por exemplo, considere o caso em que n=2 e k=2.
  - O primeiro centro é colocado exatamente sobre o ponto médio entre os dois pontos (r(C) = dist(s1,s2)/2)
  - A colocação do segundo centro, aonde quer que seja, não diminui o raio máximo
- Claramente, a distribuição dos centros no exemplo acima não é ótima
  - É possível obter r(C)=0, colocando os centros exatamente sobre os pontos
- O mesmo aconteceria para n > 2, caso os pontos estivessem distribuídos ao redor de dois pontos ‘centrais’ (houvesse dois clusters ‘naturais’)
  - Nesse caso, o algoritmo novamente posicionaria o primeiro centro entre os dois clusters; e, em seguida, não teria como diminuir o raio máximo

O problema dos k-centros (clustering)

- Agora suponha que soubéssemos que o raio máximo da solução ótima C* fosse r(C*) = r
- Nesse caso, poderíamos usar esse dado para construirmos uma solução que fosse, no máximo, 2x pior que a ótima
- O algoritmo seria:
  - Seja S’ = S; C = ∅
  - Enquanto S’ ≠ ∅
    - Selecione arbitrariamente um ponto s ∈ S’ e coloque-o em C
    - Remova de S’ todos os pontos que estiverem a uma distância máxima de 2r de s
  - Se |C| ≤ k, retorne C
  - Senão, não há solução

O problema dos k-centros (clustering)

- O algoritmo executa em tempo $O(n^2)$
- Claramente, se o algoritmo retorna um agrupamento C, então $r(C) \leq 2r$
- Lema: Suponha que o algoritmo encontre mais que k centros. Então, para qualquer solução C* com até k centros, $r(C^*) > r$.
- Prova (ideia): Prova por contradição.
  - Suponha que exista uma solução ótima C* com até k centros com $r(C^*) \leq r$
  - Como a solução C encontrada pelo algoritmo contém somente pontos de S, então, para cada $c \in C$, deve existir um $c^* \in C^*$ tal que $dist(c,c^*) \leq r$
  - Como $|C| > k$, deve existir $c' \in C$, $c' \neq c$, tal que $dist(c',c^*) \leq r$ (deve existir um outro centro que também pertence a c* na solução ótima)
  - No entanto, isso contradiz a hipótese de que dist é uma métrica, já que teríamos:
    - $dist(c,c^*) + dist(c^*,c') \leq 2r < dist(c,c')$ (por construção, c e c' devem estar a uma distância maior que 2r)
  - Portanto, c* pode ser próximo de, no máximo, um centro de C. Isso implica que $|C^*| \geq |C| > k$.

O problema dos k-centros (clustering)

- Embora seja pouco realista supor que conheçamos a solução ótima, conceitualmente, tal suposição auxilia no desenho do algoritmo de aproximação
- A partir do desenho de tal algoritmo, podemos derivar um segundo algoritmo que alcança uma garantia de desempenho similar através de tentativas para adivinhar o valor ótimo dentro de um intervalo de valores possíveis
- O algoritmo efetua uma sequência de tentativas, refinando, a cada iteração, a estimativa do valor ótimo
  - Depois de um certo número de iterações, o algoritmo converge e, consequentemente, o fator de aproximação obtido no caso em que supúnhamos saber o valor ótimo é alcançado

O problema dos k-centros (clustering)

- No caso do problema dos k centros, poderíamos adotar a seguinte estratégia:
  - Sabemos, inicialmente, que o intervalo [0,rmax] contém o valor ótimo do raio; rmax = max dist(s_i,s_j)
  - Assim, podemos usar o algoritmo anterior para determinar se é possível agrupar os pontos com r= rmax/2
    - Se for possível, reduzimos o intervalo para [0,rmax/2] e repetimos o processo como em uma busca binária
    - Caso não seja possível, procedemos de forma análoga
  - Seguimos refinando os intervalos até que os limites inferior e superior sejam relativamente próximos. Nesse momento, o fator de aproximação do algoritmo será próximo de 2

O problema dos k-centros (clustering)

- Para o caso particular do problema dos k centros, não precisamos utilizar a ‘busca binária’ para obter um fator de aproximação 2
- De fato, fazendo uma pequena alteração na forma com que os centros são escolhidos, podemos ignorar completamente o valor ótimo
- A ideia do algoritmo é a seguinte:
  - Se $k \geq |S|$, retorne S
  - Senão, selecione s arbitrário e crie $C=\{s\}$
  - Enquanto $|C| < k$
    - Selecione s que maximize dist(s,C)
    - Adicione s a C
  - Retorne C

O problema dos k-centros (clustering)

- Teorema: O algoritmo guloso retorna um agrupamento com raio, no máximo, 2x o ótimo
- Prova (ideia):
  - Suponha que a solução ótima seja r*; a maior distância de um ponto ao centro mais próximo é r*
  - Suponha, para fins de contradição, que a solução encontrada pelo algoritmo seja r > 2r*
  - Como os centros encontrados pelo algoritmo são pontos de S, existem k+1 pontos cuja distância entre si é, no mínimo, r (caso contrário a solução é ótima)
  - Na solução ótima, pelo menos dois desses pontos devem pertencer ao mesmo cluster
  - Logo, existirá um cluster com diâmetro d > 2r*, contradizendo a hipótese de que o raio ótimo é r*
  - Portanto, $d/2=r \leq 2r^*$

O problema dos k-centros (clustering)

- Teorema: Não existe algoritmo com fator de aproximação menor que 2 para o problema dos k centros, salvo se P=NP.
- Prova (ideia):
  - Vamos reduzir o problema do conjunto dominante (dominating set) ao de k centros
  - Relembrando: o problema de decisão do conjunto dominante consiste em determinar a existência de um subconjunto de k vértices de um grafo tal que todos os vértices ou pertencem ao conjunto ou são adjacentes a algum pertencente
  - A partir de uma instância do dominating set, construímos uma do k-centros atribuindo distância 1 entre vértices adjacentes e 2 para não adjacentes no grafo original
  - Note que existe solução para o problema sse existe um agrupamento com k clusters e raio=1
  - Portanto, qualquer algoritmo de aproximação com fator de aproximação menor que 2 encontraria a solução ótima para o dominating set

Leitura

- Seções 11.1 e 11.2 (Kleinberg e Tardos)