DCC207 – Algoritmos 2

Aula 15 – Soluções aproximadas para problemas difíceis (Parte 3)

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- Nessa aula, veremos outros dois exemplos de algoritmos aproximativos para problemas difíceis
- Esses dois exemplos são representantes de duas 'classes' de problemas difíceis com soluções aproximativas específicas:
    - Aqueles que só admitem soluções com fator de aproximação variável com características do problema
    - Aqueles que admitem soluções com parametrização da qualidade (e.g. $x <$ 100% do ótimo)
- Finalmente, discutiremos a hierarquia das soluções aproximativas

Set Covering

- O primeiro exemplo que veremos é de um problema que só admite solução cujo fator de aproximação é dado em função do tamanho da instância
- Relembrando: o problema de otimização do set cover consiste em encontrar o menor subconjunto de uma família de conjuntos, tal que a união desses conjuntos seja igual a união da família
- Dada uma família de conjuntos F = {S1, S2, ..., Sn}, encontrar
  - $\min \{ |C| \mid C \subseteq F \land \bigcup C = \bigcup F \}$

Set Covering

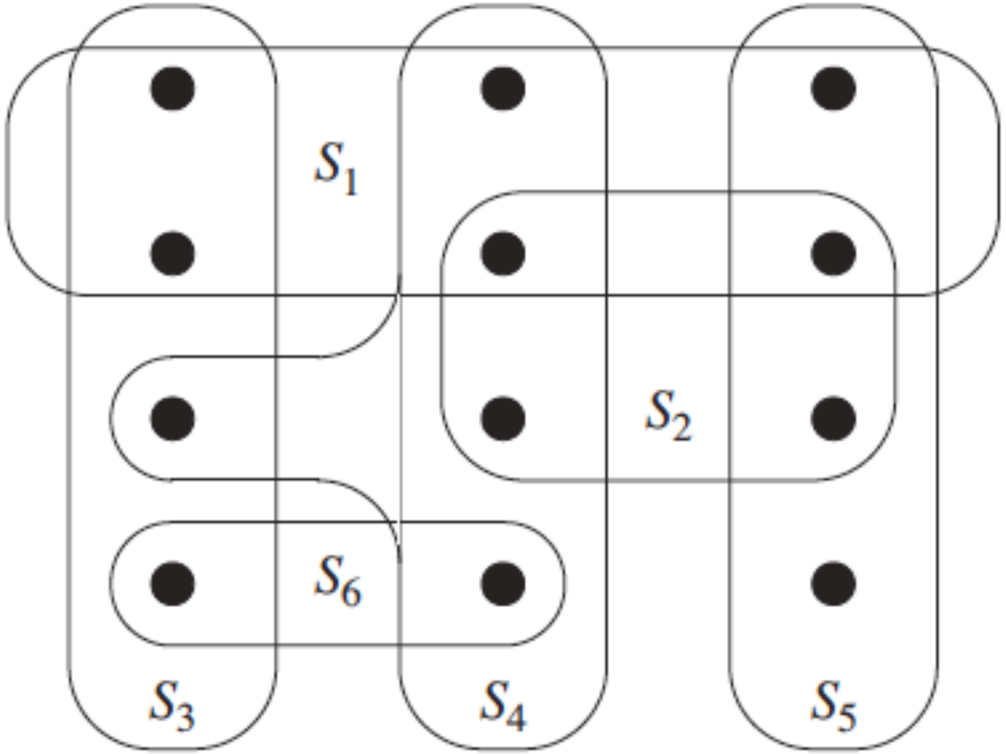
- Podemos propor uma solução gulosa para o problema da seguinte forma
- Considere X = ⋃F

GREEDY-SET-COVER$(X, \mathcal{F})$

$U = X$
$\mathcal{C} = \emptyset$
while $U \neq \emptyset$
    select an $S \in \mathcal{F}$ that maximizes $|S \cap U|$
    $U = U - S$
    $\mathcal{C} = \mathcal{C} \cup \{S\}$
return $\mathcal{C}$

Set Covering

- Considerando o exemplo ao lado, o algoritmo guloso encontra uma cobertura com 4 conjuntos:
  - S1, S4, S5 e S3 ou S6
- Porém, podemos notar que a cobertura mínima tem tamanho 3
  - S3, S4, S5
- Claramente o algoritmo executa em tempo polinomial
  - O laço externo tem custo O(min(|X|,|F|))
  - A linha 4 domina o custo interno com custo O(|X||F|)
  - Custo total O(|X||F| min(|X|,|F|))
- Qual o fator de aproximação do algoritmo?



Set Covering

- Teorema: O algoritmo guloso para set covering tem fator de aproximação $\ln |X| + 1$
- Prova (ideia):
  - Seja o d-ésimo número harmônico denotado por $H(d) = \sum^{d} 1/i$; $H(0) = 0$.
  - Vamos atribuir um custo de 1 para a escolha de cada conjunto feita pelo algoritmo
  - Como queremos escolher a menor quantidade de conjuntos, e o algoritmo escolhe conjuntos que cobrem a maior quantidade de elementos descobertos, vamos distribuir o custo da escolha do conjunto entre seus elementos que ainda não haviam sido cobertos
  - Depois vamos usar uma estimativa máxima de custo atribuído aos elementos de X para estabelecer o fator de aproximação do algoritmo

Set covering

- Seja S_i a i-ésima escolha feita pelo algoritmo guloso
- Como distribuímos o custo da escolha uniformemente entre os elementos de S_i cobertos pela primeira vez, temos que, para todo x coberto pela primeira vez, o custo de x é
  - $c_x = 1/|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})|$
- Como a escolha de cada conjunto resulta na distribuição de uma unidade entre os elementos cobertos naquela iteração, temos que
  - $|C| = \Sigma\, c_x$
- Sabemos também que os elementos de X devem estar em, pelo menos, um conjunto selecionado pelo algoritmo. Logo
  - $\sum_{S \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x = |C|$

Set Covering

- Suponha, por um instante (vamos demonstrar em breve isso), que, para qualquer conjunto S ∈ F,
  - $\sum_{x \in S} c_x \leq H(|S|)$
- Suponha que C* seja a cobertura ótima para o problema
- Portanto, das desigualdades anteriores temos:
  - $|C| \leq \sum_{S \in C^*} H(|S|) \leq |C^*| * H\big(\max\big\{|S|\big|\ S \in F\big\}\big) \leq |C^*| * H(|X|) \leq |C^*| * (\ln|X| + 1)$
  - $\sum \frac{1}{i} \leq \int \frac{1}{i} di$
- Vamos retornar agora para a demonstração de que $\sum_{x \in S} c_x \leq H(|S|)$

Set Covering

- Seja S um conjunto arbitrário de F, e $1 \leq i \leq |C|$.
- Vamos definir $u_i = |S - (S_1 \cup S_2 \cup ... \cup S_i)|$ como a quantidade de elementos de S ainda descobertos após a seleção dos i primeiros conjuntos
  - $u_0 = |S|$
- Seja k o menor inteiro tal que os k primeiros conjuntos escolhidos cobrem todos os elementos de S
  - $u_k = 0$
  - $u_{k-1} > 0$
- Logo, sabemos que:
  - $u_{i-1} \geq u_i$
  - $u_{i-1} - u_i$ elementos de S são cobertos pela primeira vez por S_i
- As deduções acima permitem concluir que
  - $\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) * \frac{1}{|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$

Set Covering

- Como o algoritmo guloso escolhe sempre o conjunto com o maior número de elementos descobertos no momento, temos que
  - $|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})| \geq |S - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})| = u_{i-1}$
- Assim
  - $\sum_{x \in S} c_x \leq \sum_{i=1}^{k} (u_{i-1} - u_i) * \frac{1}{u_{i-1}}$

$$
\begin{aligned}
&= \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}} \\
&\leq \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \\
&= \sum_{i=1}^{k} \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right) = \sum_{i=1}^{k} (H(u_{i-1}) - H(u_i)) = H(u_0) - H(u_k) = H(u_0) = H(|S|)
\end{aligned}
$$

Problema da mochila

- Como argumentado anteriormente, em diversas ocasiões, deseja-se obter uma solução que seja no máximo X% pior que a ótima
- Existem algoritmos de aproximação em que esse valor X também se torna um parâmetro
- Pode-se entender que a escolha do parâmetro X resulta em um ‘novo’ algoritmo com fator de aproximação X
- Logo, os algoritmos de aproximação que admitem esse tipo de parâmetro são conhecidos como ***esquemas de aproximação de tempo polinomial*** (**PTAS** – polynomial time approximation scheme)
- O tempo de execução desses algoritmos depende tanto do tamanho da entrada n quanto do parâmetro de qualidade ε
  - Quanto menor for o parâmetro de qualidade, maior o tempo de execução
  - A dependência de ε pode ser inclusive exponencial
  - Quando ela é polinomial, o algoritmo é chamado de fully polynomial-time approximation scheme (FPTAS ou FPAS)
- Veremos um algoritmo FPTAS para o problema da mochila

Problema da mochila

- O problema da mochila binário admite uma solução pseudo-polinomial com tempo O(nW), se o peso W não for grande
- Como visto em Algoritmos 1, podemos utilizar uma formulação de programação dinâmica para resolver o problema
- Nessa formulação, respondíamos à pergunta: qual o valor máximo que podemos obter com os k primeiros itens dentro do peso admitido?
- Podemos reformular o problema em termos do valor para encontrar uma solução com tempo O(nV), onde V=sum(vk)

Problema da mochila

- Nessa nova formulação, queremos responder à seguinte pergunta:
  - Qual é o menor peso necessário para se obter uma soma total de valores V com os k primeiros itens?
- Se $k=0$ e $V=0$, então temos que o peso é $OPT(k,V) = 0$
- Se $k=0$ e $V > 0$, então não conseguimos resolver o problema e, portanto, dizemos que o peso necessário é $OPT(k,V) = \infty$
- Se $k>0$ e $V>0$, então temos que considerar duas situações:
  - Se $v_k > V$, então não podemos usar o k-ésimo item, portanto $OPT(k,V) = OPT(k-1,V)$
  - Se $v_k \leq V$, então devemos considerar o menor peso entre $w_k+OPT(k-1,V-v_k)$ e $OPT(k-1,V)$

Problema da mochila

- Assim, temos o seguinte algoritmo:
- $V = sum(v_k)$
- DP = matriz (n+1) x (V+1)
- Para i = 1 até V: DP[0][i] = ∞
- Para k = 1 até n:
  - Para X=1 até V:
    - Se $v_k > X$, então DP[k][X] = DP[k-1][X]
    - Senão OPT[k][X] = min(DP[k-1][V], w_k + DP[k-1][X-v_k])
- Retorne max X {DP[n][X] $\leq$ W}

Problema da mochila

- Como vemos, o tempo de execução do algoritmo é dominado pelo preenchimento da matriz
- Portanto, o custo do algoritmo é $O(nV) = O(n^2 v_{max})$
- Note que, enquanto o primeiro algoritmo visto em Alg 1 era útil para situações em que o peso era pequeno, essa formulação é útil quando a soma dos valores é pequena
- Podemos tirar proveito dessa observação para construir um algoritmo aproximativo para o problema da mochila

Problema da mochila

- Podemos usar o algoritmo anterior para resolver instâncias com valores grandes, alterando a escala desses valores
- Em outras palavras, usamos uma constante $\mu = f(\varepsilon)$ para converter os valores para o múltiplo mais próximo de μ
  - Formalmente, $v_i' = \text{piso}(v_i/\mu)$
- Depois, resolvemos o problema usando o algoritmo desenvolvido
- Finalmente, reportamos a resposta do problema original como $\mu * DP(n,X)$
- Ou seja, nossa resposta terá, no pior caso, um fator de imprecisão $n\mu$ (todos arredondam para baixo)

Problema da mochila

- Mas, queremos que o algoritmo tenha fator de aproximação ε
- Sabemos que $V^* \geq v_{max}$, já que uma solução viável seria colocar o item de maior valor na mochila.
- Assim, temos que $n\mu = \varepsilon\, v_{max}$; ou seja, $\mu = (\varepsilon\, v_{max})/n$
- Pela definição v_i', também sabemos que $\mu v_i' \leq v_i \leq \mu(v_i'+1)$; e $v_i - \mu \leq \mu v_i'$
- Agora, suponha que S seja a solução retornada pelo algoritmo, e O a solução ótima

Problema da mochila

$$\begin{aligned}
\sum_{i \in S} v_i &\geq \mu \sum_{i \in S} v_i' \\
&\geq \mu \sum_{i \in O} v_i' \quad \text{(O algoritmo encontra o ótimo em V')} \\
&\geq \sum_{i \in O} v_i - |O|\mu \quad (\mu v_i' \geq v_i - \mu) \\
&\geq \sum_{i \in O} v_i - n\mu \\
&= \sum_{i \in O} v_i - \epsilon v_{max} \\
&\geq V^* - \epsilon V^* = (1 - \epsilon) V^*
\end{aligned}$$

Problema da mochila

- Em termos de complexidade de tempo, o algoritmo executa em O(nV)
- Mas, estamos usando o algoritmo sobre os valores transformados. Logo,

$$V' = \sum_{i=1}^{n} v_i' = \sum_{i=1}^{n} \left\lfloor \frac{v_i}{\epsilon v_{max}/n} \right\rfloor = O(n^2/\epsilon)$$

- Portanto, a complexidade do algoritmo se torna $O(n^3/\varepsilon)$
- Em outras palavras, o algoritmo possui tempo polinomial tanto em função de n quanto ε (ou $1/\varepsilon$)
- Note que, dessa forma, quanto menor for o parâmetro de qualidade, maior será o tempo de execução do algoritmo

Leitura

- Seções 35.3 (CLRS)
- Seção 11.8 (Kleinberg e Tardos)