Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms
  https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V
  https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Aproximação Arbitrariamente Boa

para o Problema da Mochila

- Dado um parâmetro $0 < \epsilon < 1$ (por exemplo, $\epsilon = 0.01$). Garantir uma $(1 - \epsilon)$-Aproximação.
- Parece bom demais. Entretanto o tempo de execução aumenta quando ϵ diminui.
- Pelo lado bom, podemos calibrar ϵ para uma boa troca entre qualidade de solução e tempo de execução.
- Esse é o melhor cenário para problemas NP-Difíceis em relação a aproximação.

- Para vários problemas não existe um algoritmo com aproximação arbitrariamente boa (de tempo polinomial) a menos que $P = NP$.
- Por exemplo: Vertex-Cover.

Arredondamento dos Valores dos Itens

- Ideia: Resolver de forma exata uma instância da Mochila ligeiramente incorreta, porém mais fácil que a instância original.
- Obs: Se os w_i e W são inteiros, podemos resolver a Mochila por Programação Dinâmica em tempo $O(nW)$. (note que no caso particular que W é polinomial em n, o algoritmo também é polinomial)
- Alternativamente: Se os v_i são inteiros, podemos resolver usando programação dinâmica em tempo $O(n^2 \max\{v_i\})$.

- Se todos os v_i forem pequenos (polinomiais em n), então usamos esse algoritmo para obter tempo polinomial.
- Plano: Jogar fora os bits menos significativos dos v_i's.

O algoritmo

- Passo 1: Arredondar para baixo os v_i para o múltiplo de m mais próximo. m depende de ϵ. Quanto Maior o m mais informação é jogada fora, e portanto menos acurado será a solução.

$$\hat{v}_i = \left\lfloor \frac{v_i}{m} \right\rfloor$$

- Passo 2: Resolva a mochila com valores $\hat{v}_i$, pesos w_i e capacidade W.

Knapsack: PD nos Valores

Nosso primeiro algoritmo de PD (nos pesos) para o Knapsack

- Pesos w_i e capacidade W eram inteiros.
- Tempo de execução $O(nW)$
- Uma das dimensões da tabela era W

Algoritmo de PD (nos valores) para o Knapsack

- Valores v_i são inteiros.
- Tempo de execução $O(n^2 \max\{v_i\})$
- Uma das dimensões da tabela é $n \max\{v_i\}$

- Ideia: Uma das dimensões da tabela será i que indica o prefixo $1, \ldots, i$ que é permitido usar.
- O segundo parâmetro x será o valor que desejamos obter (ou maior). E vamos procurar o menor peso que consegue obter aquele valor. $x = 0, 1, \ldots, n \cdot v_max$.
- $A[i][x]$ indicará o menor peso necessário para obter valor pelo menos x usando apenas os itens $1, \ldots, i$.

$$A[i][x] = \begin{cases} A[i-1][x] \\ w_i + A[i-1][x - v_i] \end{cases}$$

- $A[i-1][x - v_i]$ é zero se $v_i \geq x$

Algoritmo 1: KnapsackPDV(I, W)

Entrada: Um conjunto de Itens I e uma capacidade W
Saída: Valor de uma solução ótima

```
1 A[n][n max{v_i}];
2 A[0][0] = 0;
3 para x = 1, 2, ..., n max{v_i} faça A[0][x] = +∞;
4 para i = 1, 2, ..., n faça
5     para x = 1, 2, ..., n max{v_i} faça
6         A[i][x] = min{A[i-1][x]; w_i + A[i-1][x - v_i]};
7 devolva o maior x tal que A[n][x] ≤ W;
```


- Tempo de Execução $O(n^2 \max\{v_i\})$

Algoritmo $(1 - \epsilon)$-aproximado

- Passo 1: Calcular $\hat{v}_i = \lfloor \frac{v_i}{m} \rfloor$ para todo item.
- Passo 2: Resolver o problema com $\hat{v}$ usando KnapsackPDV.

Plano:

- Quão grande pode ser m, que ainda garanta uma $(1 - \epsilon)$-aproximação
- Dado esse m qual é o tempo de execução do algoritmo?

Quiz

Suponha que transformamos v_i em $\hat{v}_i$. Qual das seguintes alternativas é verdade?

a. $\hat{v}_i$ está entre $v_i - m$ e v_i
b. $\hat{v}_i$ está entre v_i e $v_i + m$
c. $m \cdot \hat{v}_i$ está entre $v_i - m$ e v_i
d. $m \cdot \hat{v}_i$ está entre $v_i - 1$ e v_i

Análise da Acurácia

Concluimos que:

- 1) $v_i \geq m \cdot \hat{v}_i$
- 2) $m \cdot \hat{v}_i \geq v_i - m$

Seja S^* a solução ótima para o problema original, e S a solução para o problema com $\hat{v}_i$. Como resolvemos de forma ótima o problema usando $\hat{v}_i$ obtemos:

- 3)

$$\sum_{i \in S} \hat{v}_i \geq \sum_{i \in S^*} \hat{v}_i$$

$$\sum_{i \in S} \hat{v}_i \geq \sum_{i \in S^*} \hat{v}_i$$

$$m \cdot \sum_{i \in S} \hat{v}_i \geq m \cdot \sum_{i \in S^*} \hat{v}_i$$

$$\sum_{i \in S} v_i \geq m \cdot \sum_{i \in S} \hat{v}_i \geq m \cdot \sum_{i \in S^*} \hat{v}_i \geq \sum_{i \in S*} (v_i - m)$$

$$\sum_{i \in S} v_i \geq (\sum_{i \in S*} v_i) - nm$$

$$\sum_{i \in S} v_i \geq (\sum_{i \in S*} v_i) - nm$$

Queremos obter

$$\sum_{i \in S} v_i \geq (1 - \epsilon) \sum_{i \in S*} v_i = \sum_{i \in S*} v_i - \epsilon \sum_{i \in S*} v_i$$

Para isso precisamos escolher um m pequeno o bastante tal que:

$$mn \leq \epsilon \sum_{i \in S*} v_i$$

$$mn \leq \epsilon \sum_{i \in S*} v_i$$

Não sabemos qual a solução ótima S^*, mas podemos pegar um m ainda menor.

$$mn = \epsilon \max\{v_i\}$$
$$m = \frac{\epsilon \max\{v_i\}}{n}$$

Algoritmo $(1-\epsilon)$-aproximado

Algoritmo 2: $(1-\epsilon)$-ApproxKnap(I, W, ϵ)

Entrada: Um conjunto de itens I, uma capacidade W e um fator ϵ
Saída: Valor de uma solução ótima

1 Calcule v_{max};
2 $m = \frac{\epsilon v_{max}}{n}$;
3 **para** $i = 1, 2, \ldots, n$ **faça**
4 $\quad \hat{v}_i = \lfloor \frac{v_i}{m} \rfloor$;
5 devolva *KnapsackPDV*(I com valores $\hat{v}, W$);

Complexidade

- Escolhendo $m = \frac{\epsilon v_{max}}{n}$ garantimos que o valor da nossa solução é $\geq (1-\epsilon)\cdot$ valor do ótimo
- Como o calculo dos $\hat{v}_i$ é linear, a complexidade total do algoritmo é a mesma do *KnapsackPDV*.

$$O(n^2 \hat{v}_{max})$$

$$\hat{v}_{max} \leq \frac{v_{max}}{m} = \frac{v_{max}}{\frac{\epsilon v_{max}}{n}} = \frac{v_{max} n}{\epsilon v_{max}} = \frac{\cancel{v_{max}} n}{\epsilon \cancel{v_{max}}} = \frac{n}{\epsilon}$$

Dessa forma a complexidade do nosso algoritmo $(1-\epsilon)$-aproximado é

$$O(n^2 \hat{v}_{max}) = O\left(n^2 \cdot \frac{n}{\epsilon}\right) = O\left(\frac{n^3}{\epsilon}\right)$$

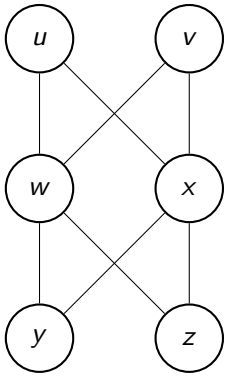
Problema do Corte Máximo

Dado um Grafo $G = (V, E)$. Encontrar um corte (A, B) (uma partição de V) que maximiza o número de arestas de corte.

- O problema do Corte Máximo é NP-difícil
- Caso particular polinomial: Grafo bipartido.

Quiz

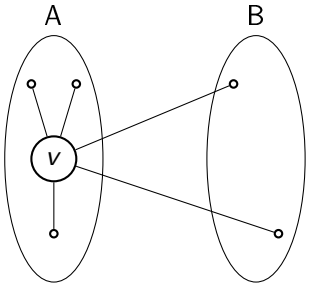
Qual é o valor do corte máximo no seguinte Grafo:



a. 4
b. 6
c. 8
d. 10

Um Algoritmo de Busca Local

- Para todo vértice $v \in V$ e um corte qualquer (A, B), definimos:
- $c_v(A, B)$ como o número de arestas incidentes em v que cruzam (A, B).
- $d_v(A, B)$ como o número de arestas incidentes em v que não cruzam (A, B)



1. Comece com um corte (A, B) qualquer.
2. Enquanto houver um vértice v com $d_v(A, B) > c_v(A, B)$:
   - move v para a outra parte.
3. devolva (A, B)

Complexidade

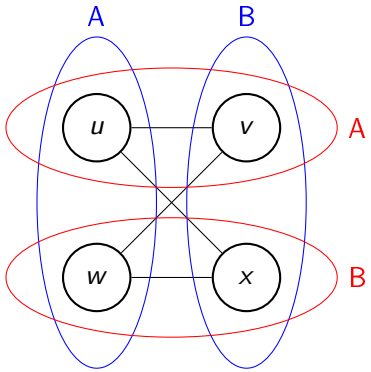
- A cada iteração do laço, o número de arestas de corte sobe em pelo menos 1.
- Dessa forma no máximo m iterações serão necessárias.
- Podemos verificar se existem vértices que atendam o critério em tempo polinomial.
- Portanto o algoritmo é polinomial.

Garantia de Qualidade

Teorema

O algoritmo de busca local apresentado, devolve um corte com o número de arestas de corte pelo menos 50% do corte máximo. (Ou ainda melhor, pelo menos 50% do número total de arestas.)

Logo provaremos esse teorema. Mas já podemos observar que essa é uma aproximação justa.



Prova

- Considere um ótimo local, ou seja, um corte em que todo vértice v atende

$$c_v(A, B) \geq d_v(A, B)$$

- Deve ser válido para a soma

$$\sum_{v \in V} c_v(A, B) \geq \sum_{v \in V} d_v(A, B)$$

- Nesse somatório cada aresta é contada duas vezes. Seja P o número de arestas que não cruzam (A, B) e Q o número de arestas que cruzam.

$$2Q \geq 2P$$
$$2Q + 2Q \geq 2P + 2Q$$
$$4Q \geq 2(|E|)$$
$$Q \geq \frac{1}{2}|E| \quad \square$$