Lecture 4: *Approximation Algorithms for Knapsack Problems*

Advanced Algorithms

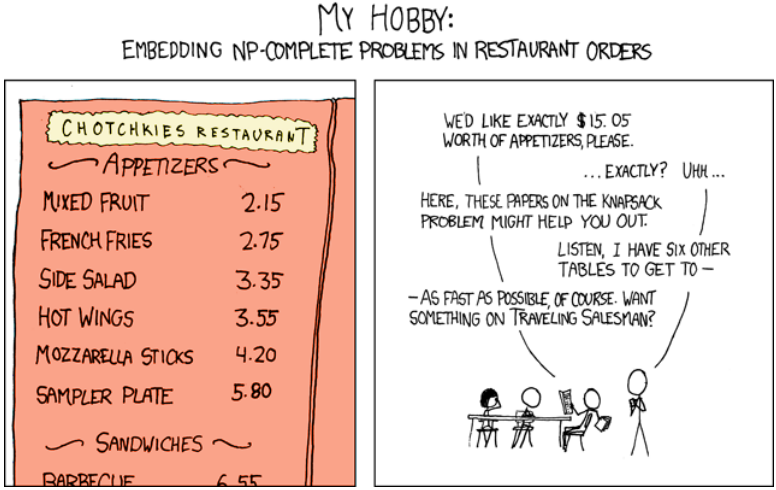
Sid Chi-Kin Chau

Australian National University

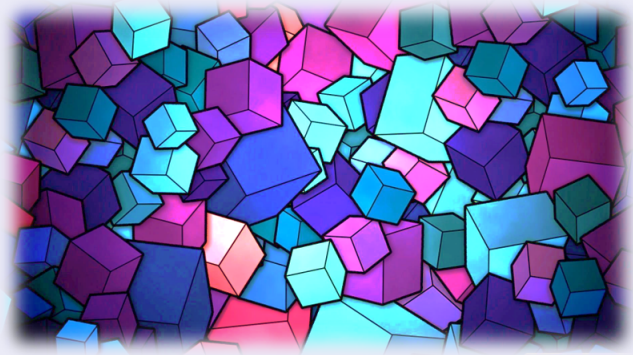
✉ sid.chau@anu.edu.au

October 5, 2022

Packing Problem is Everywhere



⋘ Packing ⋙

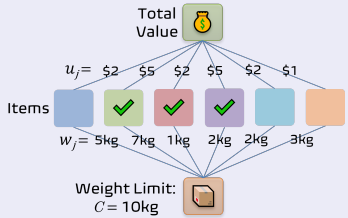


Knapsack Problem

- Knapsack problem packs a subset of items into a knapsack with a weight constraint
- Given a set of n items, the j-th item carries a value u_j and weight w_j
- We want to select a subset of items with maximum total value, subject to the total weight less than C

Definition (Knapsack)

- Maximize $\sum_{j=1}^{n} u_j x_j$
- subject to
  - $\sum_{j=1}^{n} w_j x_j \leq C$, and
  - $x_j \in \{0, 1\}$ for all $j = 1, ..., n$



NP-Completeness

Theorem (NP-Completeness)

Knapsack problem is NP-complete

Basic idea:

- Knapsack $\in$ NP-Hard
- 3SAT $\preceq$ SubsetSum $\preceq$ Knapsack

Definition (SubsetSum)

- Given n items with $\{w_j : j = 1, ..., n\}$, decide if there exists a subset of items such that
  - $\sum_{j=1}^{n} w_j x_j = C$
  - subject to $x_j \in \{0, 1\}$ for all $j = 1, ..., n$

Review: Approximation Algorithms

Definition (Approximation Ratio)

- Consider an NP-hard maximization problem $\mathcal{L}$ with an instance denoted by $\mathcal{I}$
  - Let $\mathsf{Opt}(\mathcal{I})$ be the optimal solution, and objective function be $f\big(\mathsf{Opt}(\mathcal{I})\big)$
  - Consider a polynomial-time algorithm $\mathcal{A}$ that produces a solution $\mathcal{A}(\mathcal{I})$
- Define approximation ratio: $\alpha_n(\mathcal{A}) = \min_{\mathcal{I}:|\mathcal{I}|\leq n} \dfrac{f\big(\mathcal{A}(\mathcal{I})\big)}{f\big(\mathsf{Opt}(\mathcal{I})\big)} \leq 1$

- $\alpha_n(\mathcal{A})$ is the worst-case ratio considering all instances of $\mathcal{L}$
- For NP-hard problem, $\alpha_n(\mathcal{A}) = 1$ is impossible, unless P = NP
- However, can we come up with a polynomial-time algorithm $\mathcal{A}$, such that $\alpha_n(\mathcal{A})$ is close to 1 as much as possible?

Approximation Algorithms

- Let x be a solution of polynomial-time algorithm $\mathcal{A}$ for Knapsack
- Denote $U(x) \triangleq \sum_{j=1}^{n} u_j x_j$
- Denote x^* as an optimal solution of Knapsack

Definition

- α-approximation algorithm:
  - $U(x) \geq \alpha \cdot U(x^*)$ for all inputs, for fixed constant $\alpha (< 1)$
- Polynomial-time approximation scheme (PTAS):
  - $(1-\epsilon)$-approximation algorithm for any $\epsilon > 0$
- Fully polynomial-time approximation scheme (FPTAS):
  - PTAS and additionally requires polynomial running time in $1/\epsilon$

- PTAS can give close-to-optimal solution (with exponential running time in $1/\epsilon$), but FPTAS is practically efficient (with polynomial running time in $1/\epsilon$)

Linear Programming Relaxation for Knapsack

Definition (Knapsack-LP)

- Maximize $\sum_{j=1}^{n} u_j x_j$
- subject to
  - $\sum_{j=1}^{n} w_j x_j \leq C$, and
  - $0 \leq x_j \leq 1$ for all $j = 1, ..., n$

Theorem (Optimal solution to Knapsack-LP)

- *Suppose* $\frac{u_1}{w_1} \geq \frac{u_2}{w_2} \geq ... \geq \frac{u_n}{w_n}$; *Let* $\hat{k} = \max_{k \in \{1,...,n\}} \left\{ k \mid \sum_{j=1}^{k} w_j < C \right\}$
- *Let* $x_j^{*\text{LP}} = \begin{cases} 1, & j \leq \hat{k} \\ \frac{C - \sum_{j=1}^{\hat{k}} w_j}{w_{\hat{k}+1}}, & j = \hat{k}+1 \\ 0, & j > \hat{k}+1 \end{cases}$
- $x^{*\text{LP}}$ *is an optimal solution to* Knapsack-LP

Linear Programming Relaxation for Knapsack

Proof:

- Note that $x^{*\text{LP}}$ is a feasible solution to Knapsack-LP
- Suppose $\bar{x}$ is an optimal solution, but $\bar{x} \neq x^{*\text{LP}}$
  - Note that $\sum_{j=1}^{n} \bar{x}_j w_j = \sum_{j=1}^{n} x_j^{*\text{LP}} w_j = C$
- Then there exist a pair $a < b$, such that $\bar{x}_a < x_a^{*\text{LP}} = 1$ and $\bar{x}_b > x_b^{*\text{LP}}$
- Construct a new solution x', such that $x'_j = \begin{cases} \bar{x}_j, & j \neq a, j \neq b \\ \bar{x}_a + \frac{\epsilon}{w_a}, & j = a \\ \bar{x}_b - \frac{\epsilon}{w_b}, & j = b \end{cases}$

  where ϵ is a small constant such that $x'_b \geq 0$
  - x'_j is a feasible solution (i.e. $\sum_{j=1}^{n} x'_j w_j = C$), however,

$$\sum_{j=1}^{n} x'_j u_j = \frac{\epsilon u_a}{w_a} - \frac{\epsilon u_b}{w_b} + \sum_{j=1}^{n} \bar{x}_j u_j \geq \sum_{j=1}^{n} \bar{x}_j u_j$$

    because $\frac{u_a}{w_a} \geq \frac{u_b}{w_b}$ as $a < b$
- Hence, x' is at least as good as $\bar{x}$ ($\Rightarrow$ iteratively, we can show $x^{*\text{LP}}$ is as good as $\bar{x}$)

Linear Programming Relaxation for Knapsack

Lemma (Vertex solution to Knapsack-LP)

Vertex solution x to Knapsack-LP *has at most one coordinate of x as fractional*

Proof:

- Ignore integer coordinates (i.e. $x_j = 0$ or $x_j = 1$), which do not incur tight linear constraint
- Knapsack-LP has only one linear constraint
- Hence, the rank of A is 1
- If more than one fractional coordinate x_j, then we need at least two equality constraints to uniquely determine vertex solution x
- A contradiction $\Rightarrow$ at most one coordinate x_j must be fractional

Implication:

- We can bound the gap between $x^{*\text{LP}}$ and round-off $\lfloor x^{*\text{LP}} \rfloor$

Naive Greedy Algorithm for Knapsack

Naive Algorithm

- Sort $\left\{\frac{u_j}{w_j}\right\}$ in a decreasing order (say, $\frac{u_1}{w_1} \geq \frac{u_2}{w_2} \geq ... \geq \frac{u_n}{w_n}$)
- Set $\hat{k} \leftarrow \max\left\{k = 1, ..., n \mid \sum_{j=1}^{k} w_j \leq C\right\}$
- Return solution x, where $x_j = \begin{cases} 1 & \text{if } j \leq \hat{k} \\ 0 & \text{if } j > \hat{k} \end{cases}$

Example

Consider an example with 2 items:

- $(u_1 = \frac{2}{k},\ w_1 = \frac{1}{k}), \qquad (u_2 = C,\ w_2 = C)$
- Greedy algorithm: Total value $= u_1 = \frac{2}{k}$
- Optimal solution: Total value $= u_2 = C$
- Approximation ratio $= \frac{2}{kC} \to 0$, if $k \to \infty$

Better Greedy Algorithm for Knapsack

Algorithm $\mathcal{A}_{\text{ks}}$

- Sort $\left\{\frac{u_j}{w_j}\right\}$ in a decreasing order (say, $\frac{u_1}{w_1} \geq \frac{u_2}{w_2} \geq ... \geq \frac{u_n}{w_n}$)
- Set $\hat{k} \leftarrow \max\left\{k = 1, ..., n \mid \sum_{j=1}^{k} w_j \leq C\right\}$
- Construct solution x^1, such that $x_j^1 = \begin{cases} 1 & \text{if } j \leq \hat{k} \\ 0 & \text{if } j > \hat{k} \end{cases}$
- Set $j_{\max} \leftarrow \arg\max_{j=1,..,n}\{u_j\}$
- Construct solution x^2, such that $x_j^2 = \begin{cases} 1 & \text{if } j = j_{\max} \\ 0 & \text{otherwise} \end{cases}$
- If $U(x^1) \geq u_{j_{\max}}$, then return x^1, else return x^2

- $\mathcal{A}_{\text{ks}}$ is a polynomial-time algorithm for Knapsack
- But what is its approximation ratio?

Better Greedy Algorithm for Knapsack

Theorem ($\frac{1}{2}$-approximation)

$\mathcal{A}_{\text{ks}}$ is a $\frac{1}{2}$-approximation algorithm for Knapsack

Proof:

- $U(x^{*\text{LP}}) \leq U(x^1) + u_{\hat{k}+1}$
- Also, $u_{\hat{k}+1} \leq u_{j_{\max}}$
- Hence, $U(x^{*\text{LP}}) \leq U(x^1) + u_{j_{\max}}$
- If $U(x^1) \geq u_{j_{\max}}$, then
  - $U(x^{*\text{LP}}) \leq 2U(x^1) = 2 \cdot \max\{U(x^1), u_{j_{\max}}\}$
- Else
  - $U(x^{*\text{LP}}) \leq 2u_{j_{\max}} = 2 \cdot \max\{U(x^1), u_{j_{\max}}\}$
- Let x^* be optimal integer solution for Knapsack, and noting that $U(x^*) \leq U(x^{*\text{LP}})$

Better Greedy Algorithm for Knapsack

- Is approximation ratio $\frac{1}{2}$ tight for $\mathcal{A}_{\text{ks}}$?

Example

Consider an example with 3 items:

- $u_1 = \frac{2}{k}$, $w_1 = \frac{1}{k}$
- $u_2 = \frac{C}{2}$, $w_2 = \frac{C}{2}$
- $u_3 = \frac{C}{2}$, $w_2 = \frac{C}{2}$
- Greedy algorithm: Total value $= u_1 + u_2 = \frac{2}{k} + \frac{C}{2}$
- Optimal solution: Total value $= u_2 + u_3 = C$
  item Approximation ratio $= \frac{2}{k} + \frac{1}{2} \to \frac{1}{2}$, if $k \to \infty$

PTAS for Knapsack

Algorithm $\mathcal{A}_{\text{ks2}}$ (input: ϵ)

- Set $m \leftarrow \min\{\lceil \frac{1}{\epsilon} \rceil, n\}$, set $K' = \varnothing$, set $x_j = 0$ for all j
- For every subset K of m or less items out of n items
  - Set $w(K) \leftarrow \sum_{j \in K} w_j$
  - If $w(K) \leq C$ then
    - Sort $\left\{\frac{u_j}{w_j}\right\}$ in a decreasing order (say, $\frac{u_1}{w_1} \geq \frac{u_2}{w_2} \geq ... \geq \frac{u_n}{w_n}$)
    - Set $\hat{k} \leftarrow \max\left\{k = 1, ..., n \mid \sum_{j \in \{1,...,k\} \setminus K} w_j \leq C - w(K)\right\}$
    - Construct a solution $\tilde{x}$, such that $\tilde{x}_j = \begin{cases} 1 & \text{if } j \in K \text{ or } j \leq \hat{k} \\ 0 & \text{if } j \notin K \text{ and } j > \hat{k} \end{cases}$
    - If $U(x) < U(\tilde{x})$ then set $x \leftarrow \tilde{x}$
- Return x

PTAS for Knapsack

Theorem (PTAS)

Algorithm $\mathcal{A}_{\text{ks2}}$ *is* PTAS *for* Knapsack

Basic idea:

- $\mathcal{A}_{\text{ks2}}$ uses partial exhaustive search
- Similarly, at most one coordinate in $x^{*\text{LP}}$ is fractional
- The largest to the m-largest value items in x^* are also in $\tilde{x}$
- If ϵ goes to 0, m goes to arbitrarily large
- Fix $\epsilon > 0$, then m is a constant, and $\mathcal{A}_{\text{ks2}}$ is polynomial-time in n
- Question: Is Greedy Algorithm $\mathcal{A}_{\text{ks2}}$ practical?
  - The running time is exponential in $\frac{1}{\epsilon}$

Dynamic Programming for Knapsack

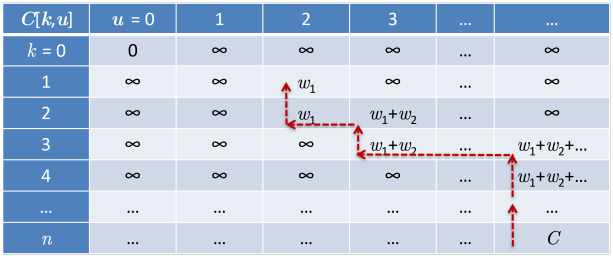
- Dynamic Programming = Divide-and-Conquer + Optimal Structure of Subproblem
- Subproblem for Knapsack:
  - Given $k \leq n$ and value u
  - Find minimum C, such that there exists a subset $R \subseteq \{1, ..., k\}$ satisfying

$$\sum_{j \in R} w_j \leq C \text{ and } \sum_{j \in R} u_j = u$$

  - If no such R exists, then $C = \infty$ (namely, not feasible for given k and u)
- Let $C[k, u]$ be the minimum capacity that is required allocate to a subset of items in $\{1, .., k\}$ which achieves a total value u
- Observe that $C[k, u] = \min \left\{ C[k-1, u], \quad w_j + C[k-1, u-u_k] \right\}$ (Bellman eqn.)
  - If $R \subseteq \{1, ..., k-1\}$ with $\sum_{j \in R} w_j \leq C$ and $\sum_{j \in R} w_j = U$, then $C[k, u] = C[k-1, u]$
  - Otherwise $C[k, u] = w_k + C[k-1, u-u_k]$

Dynamic Programming for Knapsack

- Compute $C[k, u]$ for all $k \leq n$ and $u \leq U_{\text{total}} \triangleq \sum_{j=1}^{n} u_j$ recursively
- We can find x^* by backtracking from $C[n, u] = \max\left\{C[n, u] \leq C \mid u = 1, ..., U_{\text{total}}\right\}$



- However, the input can be represented by $|\mathcal{I}| = \mathrm{O}(n \log(U_{\text{total}}))$ bits
- The running time $\mathrm{O}(nU_{\text{total}})$ is exponential in $\log(U_{\text{total}})$
- Dynamic programming is *pseudo-polynomial* time algorithm (but not polynomial-time)

FPTAS for Knapsack

- How to improve dynamic programming?
  - Rescale the number of columns in terms of n instead of U_{total}
- Let rescaling factor $\kappa \triangleq \max\left\{\frac{\epsilon \cdot \max_{j=1,\ldots,n}(u_j)}{n}, \quad 1\right\}$, hence, $\kappa \geq 1$
  - Otherwise, if $\kappa < 1$, then no need to rescale
- Let $\hat{u}_j \triangleq \lfloor \frac{u_j}{\kappa} \rfloor$ for each $j = 1, ..., n$
  - Note that $\max\limits_{j=1,\ldots,n}(\hat{u}_j) = O\left(\frac{n}{\epsilon}\right)$

Definition (RS-Knapsack (Rescaled Knapsack))

- Maximize $\sum_{j=1}^{n} \hat{u}_j x_j$
- subject to
  - $\sum_{j=1}^{n} w_j x_j \leq C$, and
  - $x_j \in \{0, 1\}$ for all $j = 1, ..., n$

- Solve RS-Knapsack by dynamic programming, as an approximation solution to Knapsack

FPTAS for Knapsack

Theorem

The running time for solving RS-Knapsack *is* $O(\frac{n^3}{\epsilon})$
The approximation ratio between RS-Knapsack *and* Knapsack *is* $(1-\epsilon)$

Proof:

- By dynamic programming, the running time is $\mathsf{O}(n \cdot n \cdot \hat{u}_j) = \mathsf{O}(\frac{n^3}{\epsilon})$
- Let $U(x) \triangleq \sum_{j=1}^{n} u_j x_j$ and $\hat{U}(x) \triangleq \sum_{j=1}^{n} \hat{u}_j x_j$
- Let x^* be an optimal solution to Knapsack, $x^{*\mathsf{RS}}$ be an optimal solution to RS-Knapsack
- Since $\frac{u_j}{\kappa} - \lfloor \frac{u_j}{\kappa} \rfloor \leq 1$, we have

$$\kappa \geq u_j - \kappa \lfloor \frac{u_j}{\kappa} \rfloor = u_j - \kappa \cdot \hat{u}_j \quad \Rightarrow \quad n \cdot \kappa \geq U(x) - \kappa \hat{U}(x)$$

$$\begin{aligned} U(x^{*\mathsf{RS}}) &\geq \kappa \hat{U}(x^{*\mathsf{RS}}) \geq \kappa \hat{U}(x^*) \geq U(x^*) - n \cdot \kappa \\ &= U(x^*) - \epsilon \cdot \max_{j=1,\ldots,n} (u_j) \geq (1-\epsilon) \cdot U(x^*) \end{aligned}$$

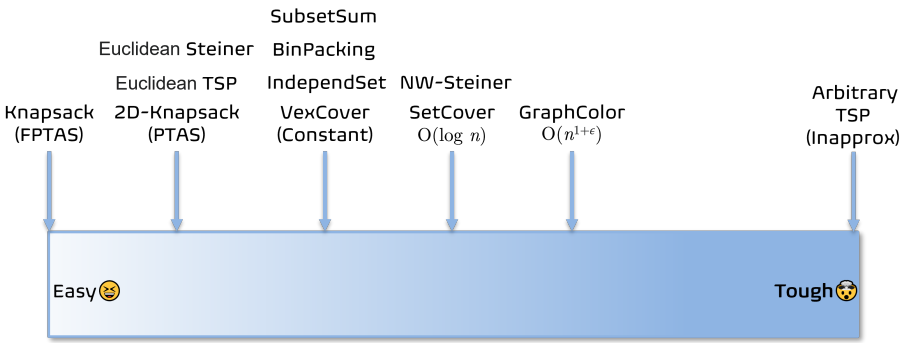
Complex-demand Knapsack

Definition (Complex-demand Knapsack)

- Maximize $\sum_{j=1}^{n} u_j x_j$
- subject to
  - $\left(\sum_{j=1}^{n} w_j^1 x_j\right)^2 + \left(\sum_{j=1}^{n} w_j^2 x_j\right)^2 \leq C^2$, and
  - $x_j \in \{0, 1\}$ for all $j = 1, ..., n$

- One can think w_j as a complex number
- We put a constraint on the magnitude of total complex-demand weight
- Complex-demand knapsack is useful for modeling non-linear power grid flow constraint
  - AC electricity power is represented by a complex number, and the magnitude of AC power is called apparent power

Spectrum of Approximability



Approximability Spectrum of NP-Complete Problems

- Why do NP-complete problems have different approximability, even though they are reducible to each other in polynomial time?
  - Polynomial-time reduction does not necessarily preserve approximation ratio
  - Worst-case instances in different problems are not equally approximable

References

Reference Materials

- Approximation Algorithms (V. Vazirani), Springer
  - Chapter 8

Recommended Materials

- Knapsack Problems (H. Kellerer, U. Pferschy, D. Pisinger), Springer
- *Combinatorial Optimization of Alternating Current Electric Power Systems* (S. Chau, K. Elbassioni, M. Khonji), Foundations and Trends in Electric Energy Systems, 2018
  `https://users.cecs.anu.edu.au/~sid.chau/FnT.html`