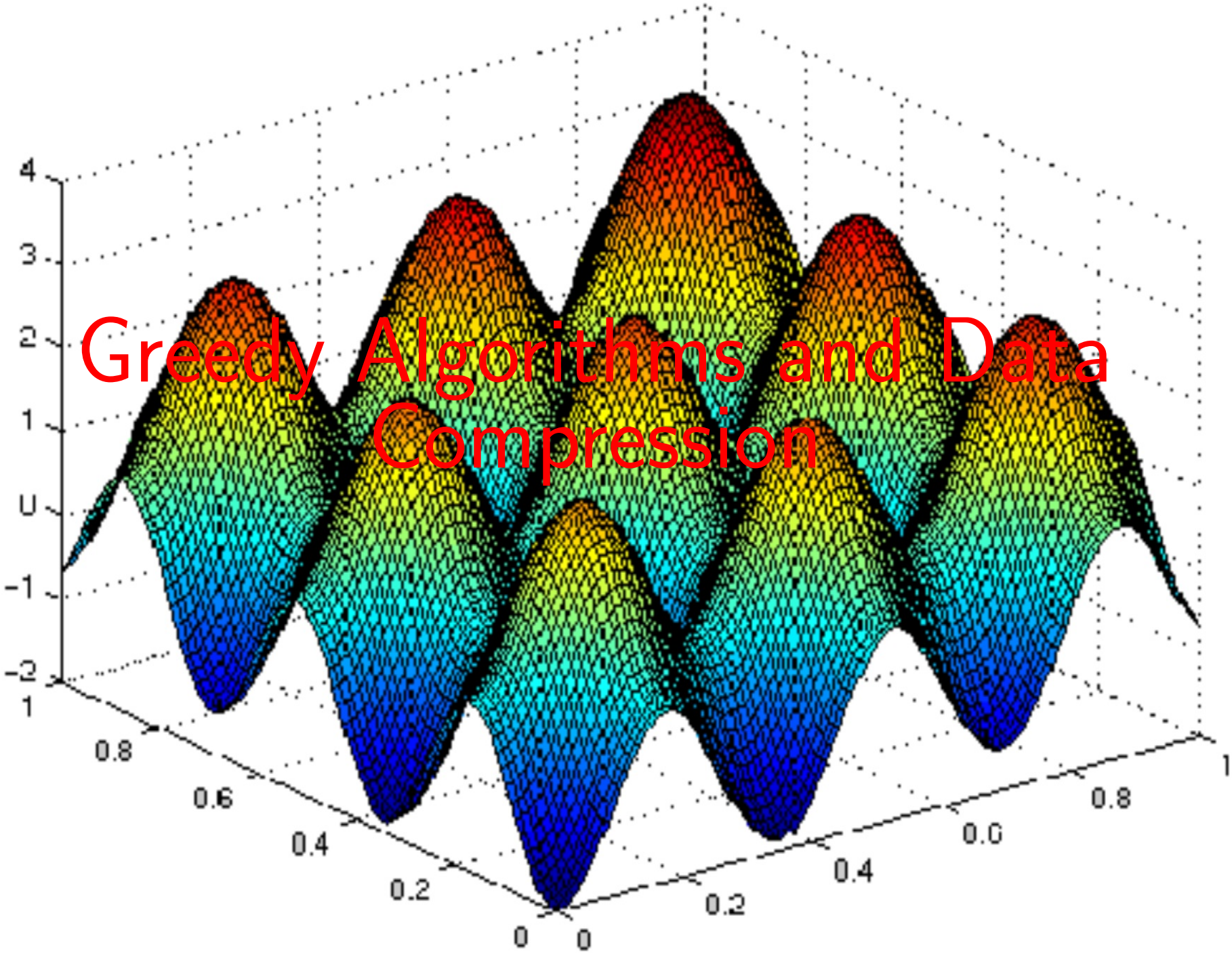
Greedy Algorithms and Data Compression



Greedy Algorithms

A greedy algorithm obtains an optimal solution to a problem by making a sequence of choices.

This technique makes a locally optimal myopic choice that leads to a globally optimal solution.

Often easy greedy algorithms are used to obtain quickly solutions to optimization problems, even though they do not always yield optimal solutions. For many problems the greedy technique yields good heuristics, or even good approximation algorithms.

Greedy Algorithms

Greedy algorithms are mainly designed for combinatorial optimization problems: Given an input, we want to compute an optimal solution according to some objective function. The solutions are formed by a sequence of elements.

A combinatorial optimization problem: Given a graph $G(V, E)$ and two vertices $u, v \in V$, we want to find a path from u to v having the minimum number of edges.

Greedy Algorithms

- At each step we choose the best (myopic) choice at the moment for the corresponding component of the solution, and then solve the subproblem that arise by taking this decision.
- The choice may depend on previous choices, but not on future choices.
- At each choice, the algorithm reduces the problem into a smaller one, and obtains one component of the solution.
- A greedy algorithm never backtracks.

Greedy Algorithms

For the greedy strategy to work correctly, it is necessary that the problem under consideration has two characteristics:

- Greedy choice property: We can arrive to the global optimum by selecting a local optimums.
- Optimal substructure: After making some local decision, it must be the case that there is an optimal solution to the problem that contains the partial solution constructed so far.

In many cases, the local criteria for selecting a part of the solution allow us to define a global order that directs the greedy algorithm.

Fractional knapsack problem

Fractional Knapsack: Given as input a set of n items, where item i has weight w_i and value v_i, together with a maximum total weight W permissible. We want to select a set of items or fractions of item, to maximize the profit, within allowed weight W.

Observe that from each item we can select any arbitrary fraction of its weight.

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w	10	20	30	40	50
v	20	30	66	40	60

Fractional knapsack:Greedy

GreedyFKnapsack (n, v, w, W)
$S = \emptyset$; $V = 0$; $i = 0$;
while $W > 0$ **do**
  Let i be the item with blabla
  if $w[i] \leq W$ **then**
    $S = S \cup \{(i, w[i])\}$; $W = W - w[i]$; $V = V + v[i]$;
  else
    $S = S \cup \{(i, W)\}$; $W = 0$; $V = V + W * v[i]/w[i]$;
  end if
  Remove i from the set of eligible items
end while
return S

GreedyFKnapsack: most valuable object

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w	10	20	30	40	50
v	20	30	66	40	60

GreedyFKnapsack: most valuable object

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w	10	20	30	40	50
v	20	30	66	40	60

Item	1	2	3	4	5
Selected	0	0	1	0.5	1

GreedyFKnapsack: most valuable object

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w	10	20	30	40	50
v	20	30	66	40	60

Item	1	2	3	4	5
Selected	0	0	1	0.5	1

Total selected weight 100 and total value 146

Correctness?

GreedyFKnapsack: the lighter object

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w:	10	20	30	40	50
v:	20	30	66	40	60

GreedyFKnapsack: the lighter object

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w:	10	20	30	40	50
v:	20	30	66	40	60

Item	1	2	3	4	5
Selected	1	1	1	1	0

GreedyFKnapsack: the lighter object

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w:	10	20	30	40	50
v:	20	30	66	40	60

Item	1	2	3	4	5
Selected	1	1	1	1	0

Total selected weight 100 and total value 156

GreedyFKnapsack: the lighter object

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w:	10	20	30	40	50
v:	20	30	66	40	60

Item	1	2	3	4	5
Selected	1	1	1	1	0

Total selected weight 100 and total value 156

Selecting the most valuable object does not provide a correct algorithm!

GreedyFKnapsack: the lighter object

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w:	10	20	30	40	50
v:	20	30	66	40	60

Item	1	2	3	4	5
Selected	1	1	1	1	0

Total selected weight 100 and total value 156

Selecting the most valuable object does not provide a correct algorithm!

Correctness?

GreedyFKnapsack: the highest ratio value/weight

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w:	10	20	30	40	50
v:	20	30	66	40	60

GreedyFKnapsack: the highest ratio value/weight

Example.

$n = 5$ and $W = 100$

Item	1	2	3	4	5
w:	10	20	30	40	50
v:	20	30	66	40	60

Item	1	2	3	4	5
ratio	2.0	1.5	2.2	1.0	1.2
Selected	1	1	1	0	0.8

Total selected weight 100 and total value 164

Selecting the lighter object does not provide a correct algorithm!

Correctness?

GreedyFKnapsack: highest ratio value/weight

Theorem

The GreedyFKnapsack selecting the item with the best ratio value/weight always finds an optimal solution to the FRACTIONAL KNAPSACK *problem*

Proof.

Assume that the n items are sorted so that

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$$

GreedyFKnapsack: highest ratio value/weight

Let $X = (x_1, \ldots, x_n)$ be the portions of items selected by the algorithm.

- If $x_i = 1$, for all i, the computed solution is optimal. We take all!
- Otherwise, let j be the smallest value for which $x_j < 1$.
- According with the algorithm,
  $x_i = 1$, for $i < j$, and
  $x_i = 0$, for $i > j$.
- Furthermore, $\sum_{i=1}^{n} x_i w_i = W$

GreedyFKnapsack: highest ratio value/weight

Let $Y = (y_1, \ldots, y_n)$ be the portions of items selected in a feasible solution,i.e.,

$$\sum_{i=1}^{n} y_i w_i \leq W$$

- We have, $\sum_{i=1}^{n} y_i w_i \leq W = \sum_{i=1}^{n} x_i w_i$
- So, $0 \leq \sum_{i=1}^{n} x_i w_i - \sum_{i=1}^{n} y_i w_i = \sum_{i=1}^{n} (x_i - y_i) w_i$
- Then, the attained values can be expressed as

$$V(X) - V(Y) = \sum_{i=1}^{n} x_i v_i - \sum_{i=1}^{n} y_i v_i = \sum_{i=1}^{n} (x_i - y_i) v_i = \sum_{i=1}^{n} (x_i - y_i) w_i \frac{v_i}{w_i}$$

GreedyFKnapsack: highest ratio value/weight

To bound the difference in values $\sum_{i=1}^{n}(x_i - y_i)w_i\frac{v_i}{w_i}$

- If $i < j$, $x_i = 1$, so $x_i - y_i \geq 0$ but as $\frac{v_i}{w_i} \geq \frac{v_j}{w_j}$,

$$(x_i - y_i)\frac{v_i}{w_i} \geq (x_i - y_i)\frac{v_j}{w_j}$$

- If $i > j$, $x_i = 0$, so $x_i - y_i \leq 0$ but as $\frac{v_i}{w_i} \leq \frac{v_j}{w_j}$,

$$(x_i - y_i)\frac{v_i}{w_i} \geq (x_i - y_i)\frac{v_j}{w_j}$$

- Pluging this in the value difference

$$V(x)-V(y) = \sum_{i=1}^{n}(x_i-y_i)w_i\frac{v_i}{w_i} \geq \sum_{i=1}^{n}(x_i-y_i)w_i\frac{v_j}{w_j} \geq \frac{v_j}{w_j}\sum_{i=1}^{n}(x_i-y_i)w_i \geq 0$$

- So, $V(X) - V(Y) \geq 0$, and x is an optimal solution.

GreedyFKnapsack: highest ratio value/weight

GreedyFKnapsack (n, v, w, W)
$S = \emptyset$; $V = 0$; $i = 0$;
while $W > 0$ **do**
  Let i be the item with higher value/weight
  if $w[i] < W$ **then**
    $S = S \cup \{(i, w[i])\}$; $W = W - w[i]$; $V = V + v[i]$;
  else
    $S = S \cup \{(i, W)\}$; $W = 0$; $V = V + W * v[i]/w[i]$;
  end if
  Remove i from the set of eligible items
end while
return S

Cost?

GreedyFKnapsack: highest ratio value/weight

GreedyFKnapsack (n, v, w, W)
$S = \emptyset$; $V = 0$; $i = 0$;
while $W > 0$ **do**
  Let i be the item with higher value/weight
  if $w[i] < W$ **then**
    $S = S \cup \{(i, w[i])\}$; $W = W - w[i]$; $V = V + v[i]$;
  else
    $S = S \cup \{(i, W)\}$; $W = 0$; $V = V + W * v[i]/w[i]$;
  end if
  Remove i from the set of eligible items
end while
return S

Cost? $O(n^2)$

GreedyFKnapsack: highest ratio value/weight

GreedyFKnapsack (n, v, w, W)
$S = \emptyset$; $V = 0$; $i = 0$;
while $W > 0$ **do**
  Let i be the item with higher value/weight
  if $w[i] < W$ **then**
    $S = S \cup \{(i, w[i])\}$; $W = W - w[i]$; $V = V + v[i]$;
  else
    $S = S \cup \{(i, W)\}$; $W = 0$; $V = V + W * v[i]/w[i]$;
  end if
  Remove i from the set of eligible items
end while
return S

Cost? $O(n^2)$ a better implementation?

Fractional knapsack

GreedyFKnapsack (n, v, w, W)
Sort the items in decreasing value of v_i/w_i
$S = \emptyset$; $V = 0$; $i = 0$;
while $W > 0$ and $i < n$ **do**
  if $w[i] < W$ **then**
    $S = S \cup \{(i, w[i])\}$; $W = W - w[i]$; $V = V + v[i]$;
  else
    $S = S \cup \{(i, W)\}$; $W = 0$; $V = V + W * v[i]/w[i]$;
  end if
  $++i$;
end while
return S

The algorithm has cost of $T(n) = O(n \log n)$.

0-1 Knapsack

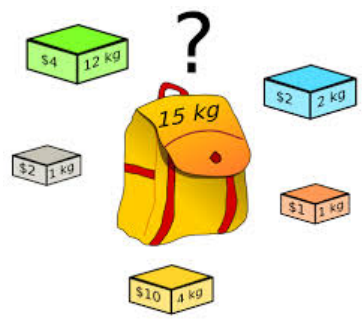
0-1 Knapsack Given as input a set of n items, where item i has weight w_i and value v_i, together with a maximum total weight W permissible. We want to select a set of items to maximize the profit, within allowed weight W.

Observe that items cannot be fractioned you take all or nothing.

The greedy algorithm for the fractional version does not work for 0-1 Knapsack

$n = 3$ and $W = 50$

Item	1	2	3
w:	10	20	30
v:	60	100	120
v/w	6	5	4



The algorithm will select item 1, which is not an optimal solution.

0-1 Knapsack

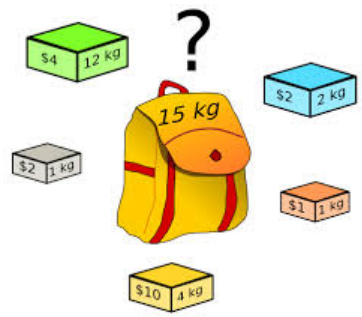
0-1 Knapsack Given as input a set of n items, where item i has weight w_i and value v_i, together with a maximum total weight W permissible. We want to select a set of items to maximize the profit, within allowed weight W.

Observe that items cannot be fractioned you take all or nothing.

The greedy algorithm for the fractional version does not work for 0-1 Knapsack

$n = 3$ and $W = 50$

Item	1	2	3
w:	10	20	30
v:	60	100	120
v/w	6	5	4



The algorithm will select item 1, which is not an optimal solution.

0-1 Knapsack is known to be NP-hard.