future solutions of subproblems.

moment is made. This choice may depend on the past, but not on
Greedy algorithms are TOP-DOWN. Whatever choice seem best at the

in general, Greedy algorithms do not give optimum solutions. In special

optimal choice

Make a choice that looks the best at the moment, i.e., locally

Greedy Approach
The optimal solution of the "big" problem can be easily modified for an optimal solution for subproblems.

- Optimal substructure property: The optimal solution of the "big" problem can be easily modified for an optimal solution for subproblems.

- Greedy-choice property: A globally optimal solution, and shows that the solution can be obtained by making a locally optimal choice. A proof examines a theorem related to "greedy" solutions:

Two major characteristics that are exhibited by most problems that lend
Optimum Solution – Find \( x_0 \) such that 
\[ \max_P x^D = P^x \]
\( x^D \) is the profit associated with \( x \).

Objective Function – \( P^x = \sum_{i=1}^{n} x_i = 1 \) and \( \sum_{i=1}^{n} x_i \geq 0 \)

Objective, such that the proportions of the objects, \( x_i \) represents the proportion of the \( i \)-th object.

Feasible Solution – A vector \( x = (x_1, \ldots, x_u) \) is feasible if \( P^x \) is the capacity of the knapsack.

Problem Instance – There are \( n \) objects. The \( i \)-th element has weight \( w_i \) and profit \( p_i \) for \( i = 1, \ldots, n \). \( W \) is the capacity of the knapsack.

Example: Rational Knapsack Problem

Greedy Approach
\[ x = (\frac{1}{2}, 0', 0') \text{ and profit } = 35. \]

Ratios are 3, 4, and 2. Objects will be arranged in order 2, 1, 3.

3. Put objects in increasing order of profit per unit of weight.

Objects are arranged in order 3, 2, and 1. \[ x = (\frac{3}{2}, 1, 0') \text{ and profit } = 34. \]

2. Put objects in increasing order of weights.

Objects are arranged in order 2, 3, and 1. \[ x = (1', 0', 0) \text{ and profit } = 30. \]

1. Put objects in decreasing order of profits.

We try three different greedy approaches:

- Optimum solution.

Given the knapsack capacity \( M = 10 \) and the

\[ \text{profit } = p_1 = 30, p_2 = 20, p_3 = 2. \]

Let \( n = 3, m_1 = 10, m_2 = 5, m_3 = 1, \text{ and the associated profits are} \]

**Knapsack Problem: Numerical Example**

Greedy Approach
Knapsack Problem: Greedy Algorithm

1. + \beta = \beta \alpha \times \alpha + d = d \ W = M, \ \frac{\beta}{M - \alpha} = \alpha \\\n
\text{Else}\ \{ x = \alpha \\\n
1 + \beta = \beta \alpha \rightarrow d \ \alpha \ M + \alpha \rightarrow \alpha \ I \ \rightarrow \beta \\\n
\text{If} \ \ W \geq \alpha \ M + \alpha \ \text{then set} \ x, \ W \geq \beta \alpha \ M \ \text{and} \ \alpha \ u \geq \beta \alpha \ M \ \text{then} \ 0 \rightarrow \beta \ I \ \cdots \ \text{set} \ x, \ \\

2. \ \text{For} \ \beta = I, \ \cdots \ \text{repeat} \ x, \ 0 \rightarrow \alpha \ I, \ 0 \rightarrow d \ I, \ \text{and the associated profit do:} \\\n
\text{the magnitude, rename them as objects, \# 1, 2, \ldots, \ n. To find the solution} x_0 \\\n
\text{After rearranging the objects by profit/weight, in decreasing order of} \\\n
\text{Knapsack Problem: Greedy Algorithm}
Proof Of Optimality

Theorem: The greedy algorithm based on strategy 3 produces the optimum solution for the rational knapsack problem.

Proof: Let $x$ and $y$ represent the greedy solutions given by the third strategy and any other solution, respectively.

Objects are renumbered in a decreasing order of the profit to weight ratio.

Suppose that $x_1 \neq y_1$. Whenever $w_1 \leq M$, $x_1 = 1$, and if $w_1 > M$, then $x_1$ is assigned maximum allowed value. Thus it is not possible that $x_1 < y_1$. 

Sanjay Ranka
The above argument will apply to the reduced problem as well. Successive objects and the knapsack capacity reduces to $m - W \cdot 1 - 1$. But in this case the problem reduces to a smaller problem of $n - 1$. Alternative solution is the optimum solution we must have $x' = \bar{x}'$. Thus, in order that the $x' \bar{D}$, proportion of the object is distributed elsewhere by $y' \bar{D}$. Since $(x' \bar{D}) \frac{(m'_1 \bar{y} - \bar{y})}{d} + \frac{(m_1 \bar{y})}{d} \bar{y} 
less \frac{(m'_1 \bar{y} - \bar{y})}{d} + \frac{(m_1 \bar{y})}{d} \bar{y} = \frac{(m_1 \bar{y})}{d} \bar{y}$.

Contribution to the first object is:

Assume that $x'_1 \bar{y} < x_1 \bar{y}$.
This.

The argument assumes that two items do not have the same profit.
Problem Instance: A single server (a processor) serves n customers. A single schedule minimizes total time in the system.

Problem Instance: A single server (a processor) serves n customers.

Example: Simulate scheduling problem (same as the Tape).
are served in increasing order of service time.

From this example it appears that in the best arrangement customers

\[ \begin{align*}
    54 & = (6+12+8)+(12+8)+8 \\
    48 & = (6+12+8)+(6+8)+8 \\
    62 & = (9+6+12)+(12)+8 \\
    54 & = (8+6+12)+(12)+8 \\
    96 & = (6+12+8)+(6+8)+8 \\
    90 & = (6+12+8)+(6+12)+8
\end{align*} \]

Total Time in Service

Order

Let \( n = 3 \), and the service times are \( s_1 = 6 \), \( s_2 = 12 \), \( s_3 = 8 \).

---

**Scheduling Problem: Example**

Greedy Approach
i.e., a permutation that is obtained by interchanging the schedule of $\mathcal{J}$.

\[
\{ u, \ldots, v, w, \ldots \} = \Pi
\]

Now consider the permutation $\mathcal{J}$. Suppose in $\Pi$ there exists a pair $j$ and $j'$ such that $j < j'$ but $j' < u$. Then

\[
\begin{align*}
&\{ u, \ldots, v, w, \ldots \} \text{ of customers} \\
&= \{ u, \ldots, v, w, \ldots \} \text{ of workers} \\
&= (u \Pi t + \cdots + v \Pi t + u \Pi t) + \cdots + (v \Pi t + u \Pi t) + v \Pi t = (\Pi) \otimes \mathcal{J}
\end{align*}
\]

Proof of Optimality

Greedy Approach


Analysis - We sort the service times and then schedule the jobs. Hence algorithms described above.

The performance of the algorithm is of the order $O(n \log n)$. If in the order $\{x, y\}$ with job $j$, repeated application of this results gives the order of job $i$ with job $j$, but $i \neq j$. Such that $i \neq j$ and $j$ improves $I$. In conclusion, a permutation of jobs schedule in which there exists a pair $i$ such that $i \neq j$ and $j$ improves $\Pi$. Thus $\Pi'$. Then $I$.

$$(\ell_{yx} - \ell_{xy})(x - j) = (\Pi)\ell - (\Pi')\ell$$

\text{and} $i$.
Objective function:

Optimum solution: A spanning tree with minimum value of the

Objective function:

Optimum solution: \( L = (L, C) \), where \( L \) is a spanning tree.

Objective function:

Optimum solution: Sum of all edges of a spanning tree

Realizable solution: A spanning tree, a collection of edges (a subset of

Problem instance: A weighted, connected, undirected graph

Example: Minimum Spanning Tree

Greedy Approach
\{e\} \cap \mathcal{L} = \varnothing
\{e\} \cap \mathcal{L} = \mathcal{L}

Suppose \{e\} denotes the next edge of \mathcal{C} in order of costs. If \{e\} \cap \mathcal{L} = \varnothing, add another edge as follows.

1. Arrange the edges by increasing order of costs.
2. Starting with the empty tree, add edges in order given above such that
3. As long as the tree size is \( |T| < n - 1 \), add another edge as follows.

Kruskal's Algorithm

Greedy Approach
We cannot have $y > z$ because the Kruskal algorithm would have added $y > z$ cost to $(\mu L)$ and $y > z$ cost to $(\nu L)$; because $y > z$

$$= (\{\nu e\} - \{\nu e\} \cup \mu L) \cap \nu L$$

forms a spanning tree and cost and choose $\nu L$ but not belong to $L$, for otherwise $L$ would have the cycle $\nu L$. Hence $\nu L$ contains a cycle $\nu L$. This cycle contains an edge $e$ that $\nu L$ does not contain $\nu e$.

Then $\nu L$ contains a cycle $\nu L$. This cycle contains an edge $e$.

Theorem. Kruskal’s algorithm generates an optimum spanning tree.

Proof. Suppose the algorithm generates a tree $L$ whose edge $e$ are $e_1, e_2, \ldots, e_{n-1}$, added in order so that $c(e_1) < c(e_2) \leq \ldots \leq (c(e_{n-1}) \leq (e_{n-1})$. Suppose $L$ is not optimum solution. Then there exists tree $L$ which is optimum MST. Suppose € is the first edge whose $L$ and $L$ differ.
Consider the attached graph:

Numerical Example:

the k-th edge before the i-th. This would imply $T_i$ is not an optimum

Greedy Approach:
Thus the algorithm proceeds as follows:

<table>
<thead>
<tr>
<th>8</th>
<th>6</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>3</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2}</td>
<td>{3, 6}</td>
<td>{2, 4}</td>
<td>{3, 5}</td>
<td>{1, 4}</td>
<td>{3, 5}</td>
<td>{1, 4}</td>
<td>{6, 7}</td>
<td>{1, 2}</td>
<td>{3, 6}</td>
<td>{2, 4}</td>
<td>{3, 5}</td>
</tr>
</tbody>
</table>

Greedy Approach

Sanjay Rana 17
edges of the same WTS.

Other WTS of equal value are also possible, because there are several

Greedy Approach
The Problem: Single-Source Shortest Paths

Example: Greedy Approach
vertices (other than \( v \)) are in \( S \). Consider the \( i \)th stage of the algorithm.

Special Path from \( s \) to a node \( v \in S - \Lambda \) to be a path whose all other edges:

\( \mathcal{W} \) denotes the weight matrix of all edges.

\( \pi \) denotes the array of parent nodes. We set \( \pi(n) = \infty \) if \( n \notin \Lambda \).

Notation: let \( d \) denote the array of shortest special paths and \( \Pi \) Denote Dijkstra's Algorithm

Greedy Approach
2. Return to step 1.

\[ \alpha = (n)^v \]

\[ (n)p = [(n', \alpha)n + (n)p', (n)p]\text{ if } (n)p = (n)^v \]

\[ [(n', \alpha)n + (n)p', (n)p]\text{ if } (n)p = (n)p \]

1.4. Fix array II: calculate new

1.3. Fix array I: For each \( n \in \mathbb{N} \), for each \( n \)

\[ \{n\} - (S - \Lambda) \rightarrow (S - \Lambda) \text{ and } \{n\} \cap S \rightarrow S \]

1.2. Update

1.1. Find node \( n \) such that \( (S - \Lambda) \ni n \) in the edge

\[ \{s\} - \Lambda \ni n \text{ for } n \in \mathbb{N} \text{ if } n \text{ is the edge } s = (n)^v \text{ and } ((n,s)m = (n)p', \{s\} = S \text{ Initial set } \]

3. The set \( (S - \Lambda) \) contains all other nodes.

2. For \( n \in S \) the minimum distance from \( s \) to \( n \) is known.

At each stage there are two sets and \( S - \Lambda \)
Consider the graph given by the following matrix. Let $s = 1$.

\[
\begin{pmatrix}
\infty & 10 & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & 50 & \infty & \infty \\
\infty & 10 & \infty & \infty & \infty & \infty \\
10 & 100 & 30 & \infty & \infty & \infty \\
\end{pmatrix}
= W
\]
$$\begin{align*}
(\mathcal{T}_0^\ast, \mathcal{E}_0^\ast) & = \mathcal{A} \\
(10, [30, 30, 30]_{\mathcal{E} = \mathcal{A}}) & = \mathcal{A} \\
(10, [\infty + 0, 30, 30, 30]_{\mathcal{E} = \mathcal{A}}) & = \mathcal{A}
\end{align*}$$
algorithm are satisfied.

shortest in their own right, in other words the conditions of a greedy
Note that a property of a shortest-path is that its sub-paths must be

\{(n', n) + (n)p, (n)p \} \text{min} = (n)p

equals the shortest path weight. This step is:

the actual minimum weight path of each vertex until the upper bound
A method that repeatedly decreases an upper bound on

Relaxation: A method that repeatedly selects a vertex \( v \) with the minimum
shortest path, inserts \( v \) into \( S \) and relaxes all edges leaving \( v \).
The algorithm repeatedly selects a vertex \( v \) with the minimum

Summary of Dijkstra's Algorithm

Greedy Approach
Therefore the algorithm is of order $O^2$.

\[
\frac{z}{(1 - u) \cdot u} = 1 + \cdots + (z - u) + (1 - u) = \text{total work}.
\]

Choosing a node of amount $(1 - u)$ — $S - \Lambda$ to put in iteration $z$.

Find a node with minimum distance $(1 - u)$ — $\Lambda$.

Initialization $(u)O^2$.

---

Analysis of Dijkstra's Algorithm

Greedy Approach
To adjust the distance of the shortest special paths, we have to look only

\[
\begin{align*}
\text{[0, 10]} & : 5 \\
\text{[4, 5]} & : 4 \\
\text{[2, 50]} & : 3 \\
\text{[2, 100, 3, 30]} & : 2 \\
\text{[2, 50, 4, 100]} & : 1 \\
\end{align*}
\]

Following lists:

direct distance to adjacent nodes. For the example graph we have the
Represent the graph by an array of \( n \) lists, one for each node, giving its
Suppose \( \# \) of edges \( u \)

**Improvement for Sparse Graphs**

Greedy Approach
nodes and a edge, time requirements are \( O(n \log n) \).

Insertion and Deletion in a heap requires \( \log n \) amount of time. For \( n \) removing the root node and replacing potentially new values of \( \langle \rangle_p(n) \) is.

So to find minimum is easy. The heap has to be maintained after maintaining a heap that contains one node of smallest values of \( S - \Lambda \).

Only for \( n \)'s that are adjacent to \( u \):

\[
(n, n) \times m + (n) p, (n) p \}
\]

\[
\text{min} = (n) p
\]

We do: \( S \)

at the adjacent nodes i.e., after finding \( u \in S - \Lambda \) that is to be added to

Greedy Approach