Coping with NP-completeness.

1. Exhaustive search. Suitable only for small problems.
2. Heuristic search. Find a solution that looks good, but without guarantee that it is optimal.
3. Approximation: Find a solution (in polynomial time) which is guaranteed to be close to optimal
4. Consider only some restricted inputs.
5. Randomization.
6. ....

Approximation Algorithms

Example: Travelling Salesman Problem.
Optimization Problem related to TSP:
Given a graph (V, E), where there are weights associated with each edge.
Problem: Find a tour (cycle) going through all the vertices (and returning to the starting vertex) with minimal weight, that is, the sum of the weights of the edges in the tour should be minimal among all such possible tours.

This problem is NP-hard. So we are unlikely to get a fast (polynomial time) algorithm to solve it.

Approximation Algorithms

Suppose the optimal tour costs \$5000, but it takes 1 year for your travel agent to find the optimal tour.

On the other hand, suppose your travel agent can find a tour costing \$5100, within 5 minutes.

For many purposes, this may be good enough.
[Also note that we may have costs associated with travel agent's time!]

So often, an approximate answer to an optimization problem may be good enough.
In this lecture we will look at some of the algorithms for "approximately" solving some NP-hard optimization problems.

Approximation algorithm for TSP when the weights satisfy triangular inequality.
If $wt(u, v) \leq wt(u, w) + wt(w, v)$, then we say that the weights satisfy triangular inequality.

Approximation Algorithm for TSP:

Input $G = (V, E)$

1. Construct a minimal spanning tree T for G.
2. Duplicate each edge in T to get a multi-graph G'.
3. Form an Euler circuit C of this graph G'.
4. While C has a vertex appearing twice {
   Suppose v appears twice in C.
   Suppose $\ldots uvw \ldots$ is the circuit.
   Change C to $\ldots uw \ldots$ (that is delete one of the v from the circuit).
   (* Note that this does not increase the weight of the circuit as $wt(u, w) \leq wt(u, v) + wt(v, w)$. *)

}

1. After step 3, we get a circuit (not necessarily simple) which goes through each vertex in the graph.
This circuit is of weight at most twice the MST and thus at most twice the optimal HC.
2. Each iteration of while loop (in step 4) does not increase the weight of the circuit.
3. At the end we have a simple circuit which goes through each vertex in the graph.
4. Thus the algorithm gives a circuit going through all the vertices of weight at most twice the weight of the optimal circuit.

Planar graph coloring

1. Whether a graph is 3-colorable is NP-hard (tutorial problem).
2. This holds even if the graph is planar.
3. There is an algorithm which colors every planar graph using 4 colors.
4. So there is an approximation algorithm which is within an additive factor 1 of optimal.

Knapsack

Instance: A set $A = \{a_1, \ldots, a_n\}$ of objects, with corresponding weights $weight(a_i)$, and values $value(a_i)$.
A knapsack of capacity K.
For a subset A' of A let,
$weight(A') = \Sigma_{a \in A'} \ weight(a)$ and
$value(A') = \Sigma_{a \in A'} \ value(a)$.

Problem: Find a subset A' of A, with $weight(A') \leq K$, which maximses $value(A')$.
Note that the knapsack problem is NP-hard

Algorithm 1:
a) Order objects in order of non-increasing value/weight.
b) In the above order, pick an object if it can fit in the remaining space ...

Algorithm 2:
a) Order objects in order of non-increasing value.
b) In the above order, pick an object if it can fit in the remaining space ...

Other ordering of objects ....

We will give an approximation algorithm for knapsack problem, which gives a solution with value at least 1/2 of optimal.
(This can be improved to be within a factor $(1 - \epsilon)$ of optimal, for any fixed $\epsilon > 0$).

We first consider an algorithm which is "almost good enough".
Input to the algorithm are:
(1) A set B of m objects, (with corresponding weights and values)
(2) A knapsack capacity K.
Output of the procedure: A subset B' of B, such that $weight(B') \leq K$.

This set B' will have "large enough" value. (The goodness of B' is not with respect to multiplicative factor but something else. More on this later).

Procedure Select

1. Sort the objects in B by non-increasing order of $value(b)/weight(b)$
   (i.e. by non-increasing order of value per unit weight).
   Let this order be $b_1, b_2, \ldots, b_m$.
2. Let spaceleft $= K$.
   $B' = \emptyset$.
3. For $i = 1$ to m do
   If $weight(b_i) \leq$spaceleft, then
   Let $B' = B' \cup \{b_i\}$.
   Let spaceleft=spaceleft$-weight(b_i)$.
   Endif
   End for
4. Output B'.

End

Lemma: Let a set of objects B (with corresponding weights and values), and knapsack capacity K be as given in the algorithm for Select. Then Select outputs a subset B' of B such that

$(\forall B'' \mid weight(B'') \leq K)$
$\quad [value(B'') - value(B') \leq \max\{value(b) \mid b \in B''\}]$
i.e. B' is worse off by value of at most "one element" of B''.

Proof:
Let B'' be an arbitrary subset of B such that $weight(B'') \leq K$.
Order the elements of B'' in order of non-increasing $value(b)/weight(b)$:
$b^1, b^2, \ldots, b^l$.

If $B'' \subseteq B'$ then we are done.

Otherwise let j be the minimum number such $b^j \notin B'$.
This implies that at the time b^j was considered in the algorithm Select, we had spaceleft $< weight(b^j)$.

Thus, $value(B') \geq$
$\Sigma_{i=1}^{j-1} value(b^i) + [K - \Sigma_{i=1}^{j-1} weight(b^i) - weight(b^j)] * [value(b^j)/weight(b^j)]$
(since at the time, when b^j couldn't have been fit, all the elements
in B' have value per unit weight more than $value(b^j)/weight(b^j)$).

Moreover, $value(B'') \leq$
$\Sigma_{i=1}^{j-1} value(b^i) + [K - \Sigma_{i=1}^{j-1} weight(b^i)] * [value(b^j)]/weight(b^j)]$,
(since we had arranged the elements of B'' in non-increasing order
of value per unit weight).

Thus,
$value(B'') - value(B') \leq weight(b^j) * [value(b^j)/weight(b^j)] =$
$value(b^j) \leq \max\{value(b) \mid b \in B''\}$. QED

Approximation Algorithm for Knapsack:

1. Use Select to get a B'.
2. Pick the best value item which can fit in the knapsack.
3. Output the better of the above two answers.
4. Now optimal is no more than the sum of answers of 1 and 2.
5. Hence, at least one of 1 and 2 above gives an answer which is atleast 1/2 the optimal.