P, NP Problems

- What is an efficient solution/algorithm?
- How to show that a problem does/does not have any efficient algorithm?
- How to show problems are as hard as another problem?
- NP-problems, NP-complete problems

Travelling Salesman Problem

- A weighted graph G (representing cities and distances between them)
- To find a tour (i.e., circuit passing through all the vertices exactly once) with minimal weight
- "Hard" to get the optimal answer
- Figuring out which problems are hard is important, so that
  - We can try near best solutions rather than optimal
  - Stop trying to find a "fast" algorithm
  - Some algorithms, such as for cryptography, rely on some problems being hard

- How do we show that some problems are hard?
- Prove lower bounds
  (for example as in sorting using comparisons)
  not always easy, specially for problems such as
  travelling salesman problem.
- Prove that it is at least as hard as some other problems,
  for which we do not know a fast algorithm
- Some of these are imporant problems, ...
- NP-complete problems

- Efficient:
- Polynomial time complexity.
- That is, for some polynomial p the time needed to solve a problem of size n is $p(n)$.

- Polynomials: are n^3, $n^5 + 6n^2$, $n \log n$ etc
- Exponential: 2^n, $2^{(n^2)}$, $2^{\sqrt{n}}$ etc.
- Superpolynomials: $n^{\log n}$.
- Note that $2^{\log n}$ is polynomial!
- However, $2^{((\log n)^2)}$ is superpolynomial (as it is $n^{\log n}$)
- Polynomials are slower growing than the exponentials.
- Though n^{100000} may not be so slow, ...

- Polynomials are closed under addition, multiplication and composition
(i.e., if $f(n)$ and $g(n)$ are polynomials, then so are $f(n) + g(n)$, $f(n) * g(n)$ and $f(g(n))$).
- If we can convert “fast” a problem A into another problem B, and there is a “fast” algorithm for B, then it reasonable to say that A can be solved fast.

Decision vs Function Problems

- Decision Problem: Yes/No answer (which can also be taken as $1/0$ answer).
- Is p a prime number?
- Is T a spanning tree of G?
- Function Problem: Problems with general output.
- Size of the smallest tour.
- Size of the longest path.

- Note that some function problems have a corresponding decision problem:
- Is the shortest tour of size smaller than 100?
- Is the shortest path of size smaller than k?
- Is the shortest path of size exactly k?
- Does the shortest path pass through vertex v?

- In some cases, answers to decision problem can be converted to answers to function problem,
- for example by checking whether the:
  1st vertex in the (lexicographically) shortest path is v_1?
  1st vertex in the (lexicographically) shortest path is v_2?
  ...
  2nd vertex in the (lexicographically) shortest path is v_1?
  2nd vertex in the (lexicographically) shortest path is v_2?
  ...
  where if there are several shortest paths, then we take the one which is lexicographically first among these.

- Function problem/optimization problem is at least as hard as the decision problem.
- We will be concentrating on decision problems only.

- P is the set of decision problems which can be solved in polynomial time in the length of the input.
- Note: We are using length of the input. Number of bits needed to write the input.
- So for example, if you use one positive integer x as input, then length of the input is $\lceil \log_2(x+1) \rceil$ (where if $x = 0$, then length of input is 1)
  We use $|x|$ to denote the length of x.
- If you have several inputs $x_1, x_2, \ldots, x_n$, then length of the input is sum of lengths of all the inputs plus n.
- most of the problems we have seen so far are in P.

- EXP is the set of all decision problems which can be solved in exponential time in the length of the input (that is in time bounded by $2^{p(n)}$ for some polynomial p).
- Clearly, $P \subseteq EXP$.
- There are problems which are not in EXP. In fact there are “arbitrarily” hard problems, even problems which cannot be solved by a computer.

NP

- NP (Informally):
  Set of problems for which 'proofs for answers' (certificates) can be verified in polynomial time.
- NP: Formally, L is in NP if there exist an algorithm A (which runs in time polynomial in the length of the input) such that
  - If $x \in L$, then for some y (called a certificate/proof) $A(x, y)$ accepts.
  - If $x \notin L$, then for all y, $A(x, y)$ rejects.
  - The length of certificate y in above cases are bounded by polynomial in the length of x (that is $|y| \leq q(|x|)$, for some fixed polynomial q).

SAT

- Input some variables $(x_1, x_2, \ldots, x_n)$ and a boolean formula over the variables such as:
  $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_4 \vee x_3) \wedge (x_1 \vee \overline{x_4})$
- Question: Is there a truth assignment to the variables such that the formula evaluates to true?

- We can solve in exponential time by considering all possible truth assignments (2^n possibilities for the truth assignment to the variables).
- Certificates: “truth assignment” which makes the formula true.
- Then we can verify if the certificates are correct, by checking if the assignment indeed evaluates to true.
- size of the certificate is polynomial (actually, $O(n)$)
- The verification process takes time $O(m)$, where m is the length of the formula.
- Thus, SAT is in NP.

- Note: If the certificate is not valid (wrong), then the verifier must reject.
- If $x \in L$: Then there is a valid/correct certificate (of length polynomial in the length of x)
- If $x \notin L$: Then there is no valid/correct certificate.
- Certificates are of length polynomial in the size of input x.
- There may be several valid certificates in case $x \in L$.

$P \subseteq NP$

- If $L \in P$, then there is a polynomial time algorithm A that can decide L.
- To show that L is in NP, the potential certificates we use is one bit.
- Verifier ignores the value of the certificate and just runs A on input x. If A accepts, then verifier also accepts. Otherwise, verifier rejects.
- Note that in the proof above, if $x \in L$, then all certificates are correct. If $x \notin L$, then all certificates are wrong.

- We can also show that $NP \subseteq EXP$
- (sketch: On a input, try all possible certificates. If one of them can be verified as correct, then accept the input. Otherwise reject the input
- Time taken is exponential, as the number of possible certificates is bounded by $2^{p(n)}$, and each certificate can be checked in polynomial time).

- Thus, $P \subseteq NP \subseteq EXP$.
- We know that $P \neq EXP$.
- We do not know whether $P = NP$ (most people think no)
- We do not know whether $NP = EXP$ (most people think no)
- If you show that $P = NP$ or $P \neq NP$, then you will become instantly famous, get Turing Award, ...

Reductions

- To show that a problem B is at least as hard as A.
- We say that $A \leq_m^p B$:
  if there is a function f which can be computed in polynomial time such that
  $x \in A$ iff $f(x) \in B$.
- Note: Hardness is modulo polynomial time. That is, if B can be solved in time $T(\text{length of input})$, then A can be solved in time
  $O(p(\text{length of input}) + T(p(\text{ length of input })))$, where p is some fixed polynomial.

- Theorem: If $A \leq_m^p B$ (as witnessed by f), and if B can be solved in polynomial time, then A can be solved in polynomial time. This can be done as follows.

  $A(x)$
  Compute $f(x)$
  Use algorithm for B to find $B(f(x))$.
  Output the answer as computed by the algorithm above.

- Time taken by the above algorithm is: $O(q(p(|x|)) + p(|x|))$, where $p(\cdot)$ is the polynomial time complexity of f, and q is the polynomial time complexity of solving B.

Some properties of redutions

- Transitive: If $A \leq_m^p B$ and $B \leq_m^p C$, then $A \leq_m^p C$.
  (If f witnesses $A \leq_m^p B$ and g witnesses $B \leq_m^p C$. Then $g(f(\cdot))$ witnesses $A \leq_m^p C$).
- Reflexive: $A \leq_m^p A$.
- If $A \leq_m^p B$, and $B \in P$, then $A \in P$.

NP Complete Problems

- L is said to be NP complete if
  1. $L \in NP$
  2. $L' \leq_m^p L$, for all $L' \in NP$.
- If condition 2 is met, then we say that L is NP-hard.
- Intuitively, NP-complete problems are the "hardest" problems in NP.
- Theorem: If L is NP-complete and L can be solved in polynomial time, then all problems in NP can be solved in polynomial time (that is $P = NP$).
- Theorem: If L_1 and L_2 are NP-complete, then both $L_1 \leq_m^p L_2$ and $L_2 \leq_m^p L_1$.
  All NP-complete problems are equally hard (easy) since each of them can be reduced to each other.

- Theorem: If B is NP-complete, $B \leq_m^p A$ and $A \in NP$, then A is NP-complete.
- Proof: Suppose $L' \in NP$.
  Since B is NP-complete, $L' \leq_m^p B$.
  Along with $B \leq_m^p A$, and transitivity of reductions we get $L' \leq_m^p A$.
  Thus, A is NP-hard.
  As $A \in NP$, we get that A is NP-complete.

To show that a problem A is NP-hard we can:

1. Directly show that it is NP-hard, by showing $L' \leq_m^p A$, for every $L' \in NP$.
2. Show that $B \leq_m^p A$, for some NP-complete problem B.

- (1) is usually hard to show.
  (2) is easier.
  But we need one NP-complete problem to start with ....

CNF formulas

- $(x_1 \vee x_2 \vee \overline{x_5}) \wedge (x_7 \vee x_3) \wedge (x_1 \vee \overline{x_4} \vee x_7 \vee x_9)$
- Variables: $x_1, x_2, \ldots$
- Literals: $x_1, x_2, \ldots, \overline{x_1}, \overline{x_2}, \ldots$
- clause: disjunction over literals.
  $(x_1 \vee \overline{x_4} \vee x_7 \vee x_9)$
- CNF: conjunction over clauses (sometimes we also say "set of clauses")
- 3-CNF formulas: each clause has exactly three literals.

SAT (satisfiability) problem

- Input: A set of variables, and a CNF over the variables.
- Question: Is there a truth assignment to the variables which makes all the clauses true?
- If yes, the CNF formula is said to be satisfiable. Otherwise it is said to be unsatisfiable.
- Example: $(x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_3)$ is satisfiable (by taking $x_1 = true$ and $x_2 = x_3 = false$)
- Example: $(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_2})$ is not satisfiable.

SAT

- SAT problem is NP-complete.
- In NP:
  1. Certificates are the truth assignments to the variables which make the formula true. Note that the length of the certificate is proportional to the number of variables, and thus linear in the length of the input.
  2. To check a certificate: just evaluate the formula on the truth assignment. It takes time proportional to the length of the formula, and thus linear in the length of the input.
- NP-Hard:
  - Proved by Cook
  - Proof a bit complicated to do in this course.

Some famous NP complete problems

1. Satisfiability:
   INSTANCE/INPUT: A set U of variables and a collection C of clauses over U.
   QUESTION: Is there a satisfying truth assignment for C?
2. 3-Dimensional Matching:
   INSTANCE/INPUT: Three disjoint finite sets X, Y, Z, each of cardinality n, and a set $S \subseteq X \times Y \times Z$.
   QUESTION: Does S contain a matching?
   That is, is there a subset $S' \subseteq S$ such that $card(S') = n$ and no two elements of S' agree in any coordinate?

3. Vertex Cover:
   INSTANCE/INPUT: A graph $G = (V, E)$ and a positive integer $K \leq card(V)$.
   QUESTION: Is there a vertex cover of size K or less for G?
   That is, is there a subset $V' \subseteq V$ such that, $card(V') \leq K$ and for each edge $(u, v) \in E$, at least one of u, v belongs to V'?

4. MAX-CUT:
   INSTANCE/INPUT: An undirected graph $G = (V, E)$, and a positive integer $K \leq card(E)$.
   QUESTION: Is there a cut of G with size $> K$?
   Here (X, Y) is said to be a cut of G, if (X, Y) is a partition of V. That is, $X \cap Y = \emptyset$ and $X \cup Y = V$. Size of a cut (X, Y) of G, is $card(\{(v, w) \mid v \in X$ and $w \in Y$ and $(v, w) \in E\})$. That is, size of a cut (X, Y) is the number of edges in G which connect X and Y.

5. Clique:
   INSTANCE/INPUT: A graph $G = (V, E)$ and a positive integer $K \leq card(V)$.
   QUESTION: Does G contain a clique of size K or more?
   That is, is there a subset $V' \subseteq V$, such that $card(V') \geq K$, and for all distinct $u, v \in V'$, $(u, v) \in E$?
6. Hamiltonian Circuit:
   INSTANCE/INPUT: A graph $G = (V, E)$
   QUESTION: Does G contain a Hamiltonian circuit?
   That is, is there a simple circuit which goes through all the vertices of G?

7. Partition:
   INSTANCE/INPUT: A finite set A and a size $s(a) > 0$, for each $a \in A$.
   QUESTION: Is there a subset A' of A such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?
8. Traveling Salesman Problem:
   INSTANCE/INPUT: A complete weighted graph $G = (V, E)$, and a bound B.
   QUESTION: Is there a Hamiltonian circuit of weight $\leq B$?

- To show $L \in NP$:
  Construct an algorithm A such that, for some polynomials p and q:

  (a) for all inputs x, y, A runs in time $p(|x| + |y|)$;

  (b) if $x \in L$: then there exists a y such that $A(x, y)$ accepts, where $|y| \leq q(|x|)$;

  (c) if $x \notin L$: then for all y, $A(x, y)$ rejects.
  y such that $A(x, y)$ accepts is called proof/certificate that $x \in L$.

  Often y above is refered to as "guess" of A on input x.
  So, A guesses the proof and verifies it.

Reduce problem A to B:

1. Define a polynomial time computable function f.
2. f maps problem instances x of A to problem instances $f(x)$ of B.
3. Show that $x \in A$ iff $f(x) \in B$.

3-SAT

- Input: a set V of variables, and a 3-CNF formula.
- Question: Is the 3-CNF formula satisfiable?
- 3-SAT is in NP:
  1. Certificates are the truth assignments to the variables. Note that the length of the certificate is proportional to the number of variables, and thus linear in the length of the input.
  2. To check a certificate: just evaluate the formula on the truth assignment. It takes time proportional to the length of the formula, and thus linear in the length of the input.

- For ease of notation, we usually represent the CNF formula (in SAT or 3-SAT) as set of clauses.
- For example,
  $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_4 \vee x_5)$.
  is represented as the set of two clauses:
  $\{(x_1 \vee \neg x_2 \vee x_3), (x_2 \vee \neg x_4 \vee x_5)\}$.
- $x_1, \neg x_2$, etc are called literals.

NP Hardness of 3-SAT

- We show $SAT \leq_m^p$ 3-SAT.
- Suppose (U, C) is an instance of satisfiability.
  We construct an instance (U', C') of 3-SAT such that, C is satisfiable iff C' is satisfiable (and the reduction can be done in poly time).
- Suppose $C = \{c_1, c_2, \ldots c_m\}$.
  For c_i, we will define C_i' and U_i' below.

$$U' = U \cup \bigcup_{1 \leq i \leq m} U_i'$$

$$C' = \bigcup_{1 \leq i \leq m} C_i'$$

- If $c_i = (l_1)$, then

  $U'_i = \{y_i^1, y_i^2\}$, and
  $C'_i = \{(l_1 \vee y_i^1 \vee y_i^2), (l_1 \vee \neg y_i^1 \vee y_i^2), (l_1 \vee y_i^1 \vee \neg y_i^2), (l_1 \vee \neg y_i^1 \vee \neg y_i^2)\}$,
  where y_i^1 and y_i^2 are NEW variables (which are not in U, and not used in any other part of the construction).

- If $c_i = (l_1 \vee l_2)$, then

  $U'_i = \{y_i^1\}$, and
  $C'_i = \{(l_1 \vee l_2 \vee y_i^1), (l_1 \vee l_2 \vee \neg y_i^1)\}$,
  where y_i^1 is NEW variable (which is not in U, and not used in any other part of the construction).

- If $c_i = (l_1 \vee l_2 \vee l_3)$, then

  $U'_i = \emptyset$, and

  $C'_i = \{c_i\}$.

- If $c_i = (l_1 \vee l_2 \vee \cdots \vee l_r)$, where $r \geq 4$, then

  $U'_i = \{y_i^1, \cdots, y_i^{r-3}\}$, and

  $C'_i = \{(l_1 \vee l_2 \vee y_i^1),$
  $(\neg y_i^1 \vee l_3 \vee y_i^2), \cdots,$
  $(\neg y_i^{p-2} \vee l_p \vee y_i^{p-1}), \cdots,$
  $(\neg y_i^{r-4} \vee l_{r-2} \vee y_i^{r-3}),$
  $(\neg y_i^{r-3} \vee l_{r-1} \vee l_r)\},$

  where $y_i^1, \cdots, y_i^{r-3}$, are NEW variables (which are not in U, and not used in any other part of the construction).

- Clearly the transformation can be done in polynomial time.
- We claim that C is satisfiable iff C' is satisfiable.
- Suppose C is satisfiable.
  Fix a satisfying assignment of C. We give a corresponding satisfying assignment of C'.
  Variables from U: same truth value as in the satisfying assignment of C.
  Other variables are given truth values as follows.
- (a) $|c_i| \leq 3$: variables in U_i' are assigned arbitrary truth value (clauses in C_i' are already satisfied).

- (b) $|c_i| > 3$:
  Suppose $c_i = (l_1, l_2, \cdots, l_r)$. Let l_j be such that l_j is true in the satisfying assignment of C fixed above.

$$(l_1 \vee l_2 \vee y_i^1), (\neg y_i^1 \vee l_3 \vee y_i^2), \cdots$$

$$\cdots (\neg y_i^{j-2} \vee l_j \vee y_i^{j-1}), \cdots$$

$$(\neg y_i^{r-4} \vee l_{r-2} \vee y_i^{r-3}), (\neg y_i^{r-3} \vee l_{r-1} \vee l_r)$$

  Then let y_i^k be true for $1 \leq k \leq j-3$, and y_i^k be false for $j-3 < k \leq r-3$. It is easy to verify that all the clauses in C_i' are satisfied.

- Now suppose C' is satisfiable. Fix a satisfying assignment of C'.
  Then we claim that the truth assignment of U' restricted to U must be a satisfying assignment for C.
- To see this suppose $c_i = (l_1 \vee \ldots \vee l_r)$.
  (a) $r \leq 3$: then c_i is clearly true due to construction.
  (b) $r > 3$:
  If y_i^{r-3} is true, then one of l_{r-1}, l_r must be true.
  If y_i^1 is false, then one of l_1, l_2 must be true.
- Otherwise pick a k such that y_i^k is true but y_i^{k+1} is false. (Note that there must exists such a k). Then l_{k+2} must be true (as we need to satisfy the clause $(\neg y_i^k \vee l_{k+2} \vee y_i^{k+1})$).

- Hence C is satisfiable iff C' is satisfiable.
  This completes the proof of 3-SAT being NP-complete.

Independent Set Problem

- Input: A graph $G = (V, E)$ and a number k.
- Question: Is there a subset $V' \subseteq V$ such that $|V'| = k$ and, for all $v, w \in V'$, $(v, w) \notin E$?
- In NP:
  - Certificates are subsets V' of V of size k with no edges among them, (clearly of size polynomial in the size of the input).
  - Verification: Check that $V' \subseteq V$ and $|V'| = k$, and for each pair v, w in V' such that $v \neq w$, check if $(v, w) \in E$;
    if none of them are in, then accept, Else reject.
- Thus, Independent Set Problem is in NP.

- NP-Hardness of Independent Set Problem:
- Reduce 3-SAT to Independent Set.
- Given a set X of variables, and a set C of clauses over X, with each clause having exactly 3-literals, form a graph $G = (V, E)$ as follows.
- Suppose the i-th clause is $(\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$.
- Let $k = |C|$.

- $V = \{u_i, v_i, w_i : 1 \leq i \leq k\}$.
- $E = P1 \cup P2$, where
  $P1 = \{(u_i, v_i), (u_i, w_i), (v_i, w_i) : 1 \leq i \leq k\}$, and
  $P2 = \{(u_i, u_j) : \ell_i^1 = \neg\ell_j^1\} \cup \{(u_i, v_j) : \ell_i^1 = \neg\ell_j^2\} \cup$
  $\{(u_i, w_j) : \ell_i^1 = \neg\ell_j^3\} \cup \{(v_i, v_j) : \ell_i^2 = \neg\ell_j^2\} \cup$
  $\{(v_i, w_j) : \ell_i^2 = \neg\ell_j^3\} \cup \{(w_i, w_j) : \ell_i^3 = \neg\ell_j^3\}$

- Intuitively, u_i represents literal ℓ_i^1, v_i represents literal ℓ_i^2, and w_i represents literal ℓ_i^3.
  In P2: “representatives” for literals x_i and $\neg x_i$ are connected.

- If the IS (independent set) problem has an independent set of size k, then (X, C) is satisfiable.
- Suppose $V' \subseteq V$ is the independent set of size k. Then note that V' must contain exactly one of u_i, v_i, w_i, for each i.
  If $u_i \in V'$ then we let l_i^1 to be true.
  If $v_i \in V'$ then we let l_i^2 to be true.
  If $w_i \in V'$ then we let l_i^3 to be true.
- Rest of the variables are assigned arbitrary truth values.
- Note that this assignment of truth values makes each clause true.

- Also this assignment of truth values is consistent as we do not make both variable x and $\neg x$ true! (as if a vertex representing x is in V', then a vertex representing $\neg x$ cannot be in V' as there will be an edge between these two vertices).

- If (X, C) is satisfiable then the IS problem has an independent set of size k.
- Suppose (X, C) is satisfiable.
- Then each clause has a true literal.
- For each i, place in V' one of the vertices from u_i, v_i, w_i such that the literal corresponding to it is true (if several of them are true, choose an arbitrary one).
- Now, the set V' is of size k.
- It is an independent set as we didn't choose two vertices from any of (u_i, v_i, w_i), thus no edge in $P1$ contains two endpoints in V'. Furthermore, no edge in $P2$ contains two endpoints in V' as these edges only connect vertices corresponding to x and $\neg x$ (for some x) both of which cannot be true.

Clique

- Input: A graph $G = (V, E)$ and a number k.
- Question: Is there a clique of size k? That is, is there a subset $V' \subseteq V$ such that $|V'| = k$, and for all $v, w \in V'$ such that $v \neq w$, $(v, w) \in E$.
- Clique is in NP:
- Certificates: Subsets V' of V of size k, where all the edges between pairs of vertices in V' is present in E.
- Verification: Given a certificate V', check whether
(a) it is of size k, subset of V and
(b) whether for all $v, w \in V'$ such that $v \neq w$, $(v, w) \in E$.

- Reduction: Independent set problem to Clique
- If $G = (V, E)$ and k is an Independent set problem, then form a Clique problem as follows:
- $G' = (V, E')$, where $E' = \{(v, w) : v \neq w, (v, w) \notin E\}$.
  Let $k' = k$
  (Intuitively, G' is complement of G)
- Then, the Clique problem is (G', k').
- It is easy to verify that $V' \subseteq V$ is an independent set in G iff V' is a clique in G'.
- Thus, G has an independent set of size k iff G' has a clique of size k.

Hamiltonian Circuit (HC)

- Given a graph $G = (V, E)$, Hamiltonian circuit of G is a simple circuit which passes through all the vertices of the graph. That is, a listing $v_1, v_2, \ldots, v_n$ of the vertices of G such that, $v_1, v_2, \ldots, v_n$ are distinct, and $(v_i, v_{i+1}) \in E$, for $1 \leq i < n$, and $(v_n, v_1) \in E$.
- It can be shown that Hamiltonian Circuit Problem is NP-complete.
- Easy to show that HC $\in$ NP.
- Proof of HC is NP-hard is very difficult and beyond the scope of this module.

Travelling Salesman Problem (TSP)

- Given a weighted graph $G = (V, E)$, with weight given by wt, and a number w, is there a Hamiltonian circuit in G with weight at most w?
- TSP is NP-complete.
- Easy to show that TSP is in NP (certificates would be Hamiltonian circuits which are of weight $\leq w$)

- NP-Hardness: Reduction from HC.
- Given $G = (V, E)$, a Hamiltonian circuit problem construct a TSP problem $G' = (V', E')$, weight matrix wt and w as follows.
- $V' = V$.
- $E' = \{(u, v) : u \neq v, u, v \in V\}$
- $wt((u, v)) = 1$, if $(u, v) \in E$.
- $wt((u, v)) = n + 1$, if $(u, v) \notin E$.
- $w = n = |V|$.

- Clearly, there are no Hamiltonian circuits in G' of weight $< n$.
- It is easy to verify that a Hamiltonian circuit in G is a Hamiltonian circuit in G' with weight n, and a Hamiltonian circuit in G' with weight n is a Hamiltonian circuit in G.
- Thus, $HC \leq_m^p TSP$.
- Hence TSP is NP-complete.

Coping with NP-completeness

- Decision version of the problems are NP-complete. Optimization versions are NP-hard. So how to handle?
- Exhaustive search. Suitable only for small problems.
- Heuristic search. Find a solution that looks good, but without guarantee that it is optimal.
- Approximation: Find a solution (in polynomial time) which is guaranteed to be close to optimal
- Consider only some restricted inputs.
- Randomization.
- Pesudopolynomial time algorithms: Polynomial in "numbers" involved rather than the length of the representation of the numbers (which is logarithmic in the numbers).
- ....

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.
- We will now look at some of the algorithms for "approximately" solving some NP-hard optimization problems.
- Absolute Error:$|opt - ValSol|$
- Relative Error (for minimization problems): $re = \frac{ValSol - opt}{opt}$
- Accuracy Ratio (performance ratio): $\frac{ValSol}{opt}$ or $\frac{opt}{ValSol}$ based on minimization/maximization problem.
- optimal value/solution may not be known: use some upper/lower bound for it.

c-approximation algorithms

- $|opt - ValSol| \leq c$, for some constant c.
- Accuracy Ratio (performance ratio): $\frac{ValSol}{opt} \leq c$ or $\frac{opt}{ValSol} \leq c$ based on minimization/maximization problem, for some constant $c > 1$.

Some Heuristics for TSP

- Start at any vertex. From the current vertex, go to the nearest unvisited vertex, until all vertices are visited.
- Do as in Kruskal's algorithm. That is:
  - Sort the edges in increasing order of weight.
  - Starting from the minimal weight edge, at each step, add the next edge if it does not create a vertex of degree three.

Is there an approximation algo for TSP?

- Theorem: If $P \neq NP$, then there is no c-approximation algorithm for TSP.
- Proof: Suppose otherwise. Then we solve HC problem for $G = (V, E)$ as follows:
- Construct $G' = (V, E')$, with E' consisting of all possible edges with weight of the edges as follows:
  $wt(e) = 1$, if $e \in E$.
  $wt(e) = (c+1)n$, if $e \notin E$.
- Now G has a HC, iff then there is a TSP for G' with weight $\leq n$ iff there is a TSP for G' with weight $\leq cn$.
- Thus, a c-approximation solution for TSP in G' will give a solution to HC problem in G.

- 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 Do
   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 $wt(u, w) \leq wt(u, v) + wt(v, w)$. *)

EndWhile

- 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.
- Each iteration of while loop (in step 4) does not increase the weight of the circuit.
- At the end we have a simple circuit which goes through each vertex in the graph.
- Thus the algorithm gives a circuit going through all the vertices of weight at most twice the weight of the optimal circuit.

Christofides Algorithm

- Instead of doubling the edges, only add the minimal weight matching for odd-degree nodes in the minimal spanning tree.
- This guarantees Eulerian Circuit in the so formed graph (that is a circuit which goes through all edges of the graph).
- Then this Eulerian Circuuit can be converted to simple circuit as in the previous algorithm.
- This gives a 1.5 factor approximation algorithm (proof not done for this class).
- There are some other algorithms such as 2-opt, 3-opt, Lin-Kernighan algorithms, which also do reasonably well in practice.

Planar graph coloring

- Whether a graph is 3-colorable is NP-hard (tutorial problem).
- This holds even if the graph is planar.
- There is a polynomial time algorithm which colors every planar graph using 4 colors.
- 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
- $(\forall B'' \mid weight(B'') \leq K)$
  $[value(B'') - value(B') \leq \max\{value(b) \mid b \in B''\}]$
  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:

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