Search Problems and Computability I

SAT-SEARCH:
Input: Set of clauses $S = \{C_1, \ldots, C_m\}$.
Output: A truth assignment t which satisfies S, or "No" if no such assignment exists.

Theorem 1 SAT-SEARCH $\xrightarrow{p}$ SAT.

Idea of the reduction:

Given $S = \{C_1, \ldots, C_m\}$, let the variables in S be $P_1, P_2, \ldots, P_k$.
Set P_1 to be T (true).
Ask if $S[P \leftarrow T] \in$ SAT?
if so, define $t(P_1) = T$;
if not, define $t(P_1) = F$.
Update S and continue with the remaining variables.

Definition 1 We define $S[P \leftarrow T]$. For each clause C_i in S, if C_i includes literal P (unnegated), then delete C_i. If C_i includes $\bar{P}$, delete $\bar{P}$ from C_i. Dually for $S[P \leftarrow F]$.

Example. $S = \{P \vee \bar{Q}, R \vee \bar{S}, \bar{R} \vee \bar{P}, R \vee Q\}$.
$S[P \leftarrow T] = \{R \vee \bar{S}, \bar{R}, R \vee Q\}$, and $S[P \leftarrow F] = \{\bar{Q}, R \vee \bar{S}, R \vee Q\}$.

To reduce SAT-SEARCH to SAT, on input S:

If $S \notin SAT$, then output "No" and halt.
Let the variables in S be $P_1, P_2, \ldots, P_k$.
for $i : 1..k$ do
if $S[P \leftarrow T] \in SAT$,
then $S \longleftarrow S[P \leftarrow T]$,
$t(P_i) \longleftarrow T$,
else $S \longleftarrow S[P \leftarrow F]$,
$t(P_i) \longleftarrow F$,
end if
end for
Final t satisfies S.

[1]These notes were taken originally by Eugenia Ternovskaia in Steven Cook's csc364 lecture, Spring 99, and they have been edited for this term.

Lemma 1 If t is an assignment such that $t(P) = T$, then t satisfies S iff t satisfies $S[P \leftarrow T]$. Similarly, if $t(P) = F$, then t satisfies S iff t satisfies $S[P \leftarrow F]$.

General Claim: Let XS be a search problem, and let XD be the corresponding decision problem, If XD is **NPC**, then $XS \xrightarrow{p} XD$.

SUBSET-SUM:
Input: $d, w_1, \ldots, w_d$, $T \in \mathbb{N}$.
Question: Is there $S \subseteq \{1, \ldots, d\}$ s.t. $\sum_{i \in S} w_i = T$?

SUBSET-SUM-SEARCH:
Input: $d, w_1, \ldots, w_d$, $T \in \mathbb{N}$.
Output: Find $S \subseteq \{1, \ldots, d\}$ s.t. $\sum_{i \in S} w_i = T$, or say "No" if no such S exists.

Theorem 2 SUBSET-SUM-SEARCH $\xrightarrow{p}$ SUBSET-SUM.

Idea of the reduction:

Input: $I =< d, w_1, \ldots, w_d, T >$.
If $I \notin$ SUBSET-SUM, output "No".
We want to find S so that $\sum_{i \in S} w_i = T$. Is $1 \in S$?
    $S \longleftarrow \emptyset$
    Ask if $< d-1, w_2, \ldots, w_d, T > \in$ SUBSET-SUM?
        if so, $I \longleftarrow < d-1, w_2, \ldots, w_d, T >$;   % i.e., throw away w_1.
        if not, $S \longleftarrow S \cup \{1\}$.
            $I \longleftarrow < d-1, w_2, \ldots, w_d, T - w_1 >$.
Continue with w_2, w_3, etc.

PARTITION:
Input: $d, w_1, \ldots, w_d$.
Question: Is there $S \subseteq \{1, \ldots, d\}$ s.t. $\sum_{i \in S} w_i = \sum_{i \notin S} w_i$?

PARTITION-SEARCH:
Input: $d, w_1, \ldots, w_d$.
Output: Find $S \subseteq \{1, \ldots, d\}$ s.t. $\sum_{i \in S} w_i = \sum_{i \notin S} w_i$.

Theorem 3 PARTITION-SEARCH $\xrightarrow{p}$ PARTITION.

The proof of this is question 2 in Problem Set 3.

Graph Colouring

Let C be a finite set (of colours). Let $G = (V, E)$.

Definition 2 A *C-colouring* of G assigns a member of C (i.e., a colour) to each vertex in G so that adjacent vertices get different colours.

k-COLOUR: (Here k is a positive integer constant.)
Input: $G = (V, E)$.
Question: Can G be coloured with k colours?

Theorem 4 3-COLOUR is **NPC**.

To prove the theorem, we need to show that 3-COLOUR is **NP**-hard. We do it by a reduction from 3SAT. The reduction is described in Cormen et al, p. 962.

Exercise: Show that 3-COLOUR $\leq_p$ k-COLOUR if $k \geq 3$. So, k-COLOUR is **NPC** for $k \geq 3$. Also, show that 2-COLOUR$\in$ **P**.

Computability

Readings: Sipser, Chapter 3. In this part of the course, we will

- Define the notion of an algorithm (TM).
- Prove that some problems (languages) cannot be solved by any algorithm.

An example of a problem that cannot be solved by any algorithm is the Halting problem: given a program P and an input x to P, decide whether P halts on input x.

Turing (1936): Introduced TMs. Proved no TM can solve the Halting problem. Argued that any algorithm can be simulated by a TM. This statement in often called the Church-Turing Thesis.

Note that TMs in Sipser have one-way input tape. If a TM M scans the left end of its tape, and it is told to shift left, it does not move.

Claim 1 Sipser's TMs and our TMs can simulate each other.

How does a TM with one-way tape can simulate a TM with two-way tape?

Idea 1: Use markers for left end of the tape and for the right end of the used tape space. If scanning the left end and want to shift left, move the entire tape content right one square.

Idea 2: "Fold" the tape. Store the top part of the tape on even squares, bottom part on odd squares.

General Claim: A TM can simulate every reasonable variation of a TM (e.g. a TM with several tapes).

Let $L(M)$ be the language accepted by a TM M. Then $x \in L(M)$ iff M, when started on input x, eventually enters the state q_{accept}. We assume that the input x is in Σ^*, so $L(M) \subseteq \Sigma^*$.

Note that x can fail to be in $L(M)$ for two reasons: Either M fails to halt on input x, or M halts in the reject state.

Definition 3 Let $L \subseteq \Sigma^*$. Then L is *semi-decidable* if $L = L(M)$ for some TM M. (Note that Sipser says "recognizable" instead of "semi-decidable".)

Definition 4 A TM M *decides* L if $L = L(M)$ and M halts for every input $x \in \Sigma^*$.

Definition 5 L is *decidable* if some TM decides L.

Example: The Halting problem is semi-decidable, but not decidable.

Definition 6 An *enumerator* E is a TM with a separate write-only output tape. (Sipser says printer instead of output tape.)

E writes strings on its output. $L(E)$ is the set of strings x such that when E starts with a blank tape, x is eventually written on the output. Thus $L(E) \subseteq \Gamma^*$, where Γ is the output alphabet.

Definition 7 L is *enumerable* iff $L = L(E)$ for some enumerator E.

Theorem 5 L is enumerable iff L is semi-decidable.