Search and Optimization Problems

These notes supplement the old CSC 364S course notes "**NP and NP-Completeness**" and "**Turing Machines and Reductions**" by presenting **NP** Search and Optimization problems.

Problems in **NP** are formally sets of strings, but we often define them as *decision problems*. For example **SAT** is defined as follows:

SAT
Instance:
$\langle\varphi\rangle$, where φ is a formula of the propositional calculus.

Question:
Is φ satisfiable?

Thus **SAT** is the problem: given a propositional formula, decide whether or not it is satisfiable. But in practice we often want to know more: If φ is satisfiable, we would like to find a satisfying truth assignment. This problem can be stated as follows:

SAT-SEARCH
Instance:
$\langle\varphi\rangle$, where φ is a formula of the propositional calculus.

Output:
A satisfying assignment for φ, or 'NO' if none exists.

This idea can be generalized to apply to arbitrary sets $A \subseteq \Sigma^*$ in **NP**. By definition (see **Definition 1** in the notes **NP and NP-Compltenes**) A is in **NP** iff there is a polynomial time computable relation $R(x, y)$ and constants c, d such that for all $x \in \Sigma^*$

$$x \in A \Leftrightarrow \text{there exists } y \in \Sigma^* \text{ so } |y| \leq c|x|^d \text{ and } R(x, y)$$

Here we call any string y a *certificate* for x if it satisfies the conditions $|y| \leq c|x|^d$ and $R(x, y)$ in the definition.

The corresponding search problem for A is

A-SEARCH
Instance:
$x \in \Sigma^*$

Output:
$y \in \Sigma^*$ such that $|y| \le c|x|^d$ and $R(x, y)$, or 'NO' if no such y exists.

It turns out that if A is **NP**-complete, then the two problems A (the decision problem) and **A-SEARCH** are polynomial time reducible to each other.

For this kind of polynomial reducibility we refer the reader to Definition 6 in the Notes **Turing Machines and Reductions**. Repeating this definition we have

Definition 6. P_1 is polynomial-time reducible to P_2 (in symbols: $(P_1 \xrightarrow{p} P_2)$) if there is a polynomial-time algorithm for P_1 which is allowed to access a solver for P_2, where the time taken by P_2 is not counted.

Theorem 1 (Self Reducibility)**.** *1) If A is any problem in* **NP***, then $A \xrightarrow{p}$ A-SEARCH.*

2) If A is **NP-complete** *then A-SEARCH $\xrightarrow{p} A$.*

Proof. The proof of 1) is obvious: An input x is in A iff the answer to **A-SEARCH** is a certificate y for x.

For the proof of 2), we use the fact that if A is **NP-complete**, then *every* decision problem B in **NP** is polytime reducible to A. We leave it to the reader to think of a useful **NP** problem B such that the answers to polynomially many queries to B can be used to find a certificate y for x (assuming $x \in A$). □

Although we know from part 2) of the above theorem that A-SEARCH $\xrightarrow{p} A$ when A is **NP**-complete, it is interesting to give explicit reductions from search to decision for specific **NP-complete** problems A.

Example 1: **SAT-SEARCH** $\xrightarrow{p}$ **SAT**. (i.e. **SAT** is self-reducible.)

Proof. Assume that $Sat(\varphi)$ is a Boolean solver for **SAT**. Thus

$$Sat(\varphi) \text{ is true } \Leftrightarrow \varphi \in \textbf{SAT}$$

We assume that Boolean formulas can have constants 1 (for true) and 0 (for false). We use the notation $\psi[x_i \leftarrow 1]$ for the result of replacing every instance of the variable x_i in formula ψ by 1, and similarly for $\psi[x_i \leftarrow 0]$.

Below is the program: (We assume that the input formula φ has variables $x_1, \ldots, x_n$.)

Input φ
if $\neg Sat(\varphi)$ then output 'NO'
$\psi \leftarrow \varphi$
for $i = 1 \ldots n$ (*)
    if $Sat(\psi[x_i \leftarrow 1])$ then
        $\psi \leftarrow \psi[x_i \leftarrow 1]; \tau(x_i) = 1$
    else $\psi \leftarrow \psi[x_i \leftarrow 0]; \tau(x_i) = 0$
    end if
end for
Output τ

(*) Loop Invariant: ψ is satisfiable and $\psi = \varphi[x_1 \leftarrow \tau(x_1), \ldots, x_i \leftarrow \tau(x_i)]$.

$\square$

Example 2: Recall that if $G = (V, E)$ is an undirected graph and $V' \subseteq V$, then V' is a *clique* in G iff $(u, v) \in E$ for every pair u, v of distinct nodes in V'. The associated decision problem is:

CLIQUE
Instance:
$\langle G, k\rangle$ where G is an undirected graph an k is a positive integer.

Question:
Does G have a clique of size k?

The associated search problem for the same input as above is to find a clique of size k, if one exists. But a more interesting associated search problem is the following **optimization** problem:

MAX CLIQUE-SEARCH
Instance:
$\langle G\rangle$ where $G = (V, E)$ is an undirected graph.

Output:
A clique $V' \subseteq V$ in G such that $|V'| \geq |V''|$ for every clique V'' in G.

Theorem 2. MAX CLIQUE-SEARCH $\xrightarrow{p}$ CLIQUE.

Proof. Assume that $Clique(G, k)$ is a Boolean solver for **CLIQUE**. The program for **MAX CLIQUE-SEARCH** has two parts. On input $G = (V, E)$, the first part finds the largest number k_G such that G has a clique of size k_G, and the second part finds a clique of size k_G.

Here is the program for **MAX CLIQUE-SEARCH**. We assume that the input graph is $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$.

If H is a graph, then the notation $H - \{v_i\}$ stands for the graph obtained from H by removing the vertex v_i and all edges incident to v_i.

for $i = 1 \ldots n$
    if $Clique(G, i)$ then $k \leftarrow i$
end for
$k_G \leftarrow k$

$H \leftarrow G$
for $i = 1 \ldots, n$ (*)
    if $Clique(H - \{v_i\},\ k_G)$ then $H \leftarrow H - \{v_i\}$
end for
V' = the set of vertices in H.
Output V'

Correctness proof:

It is clear from the first part of the program that k_G is the size of the largest clique in G.

To see that the output V' of the second part is a clique of size k_G we use the following loop invariant (which is proved by induction on i):

(*) Loop invariant:

Let $H = (V_i, E_i)$. Then H has a clique V' of size k_G, where

$$V_i \cap \{v_1, \ldots, v_{i-1}\} \subseteq V' \subseteq V_i$$

Hence after the for loop is finished, in effect the next $i = n + 1$, so $V_{n+1} = V'$, where V_{n+1} is the set of vertices in the final graph H. Thus the set of vertices in the final H is a clique of size k_G. $\square$