CSE 200 Lecture 10/17/06

Chris Calabro

October 26, 2006

1 Self reducibility

An optimization problem typically involves specifying a set of inputs, a set of outputs, a 2-ary objective function f, a 2-ary predicate constraint C, and, at least implicitly, whether we are trying to maximize or minimize. (Of course one can be converted to the other by replacing the objective function by its negative. Wlog we will assume all problems are maximization problems.) Usually the set of inputs, the set of outputs are decidable in polytime, and f, C are computable in polytime. A problem *instance* is an input x. A *feasible solution* is a y satisfying $C(x, y)$. The problem can be formulated in multiple ways:

decision also given $k \in \mathbb{Z}$ as input, decide whether there is a feasible y with $f(x, y) \geq k$.

search also given $k \in \mathbb{Z}$ as input, find a feasible y with $f(x, y) \geq k$.

opt find the maximum of $f(x, y)$ over all feasible y.

search opt find a feasible y maximizing $f(x, y)$.

1.1 Example: maximum independent set

The maximum independent set problem is a maximization problem with a graph G as input, a subset S of the nodes as output, $|S|$ as the objective function, and $\forall x, y \in S\ \{x, y\} \notin E(G)$ as the constraint. A feasible S is also called an *independent set*. The decision, search, opt, search opt problems are therefore

decision also given $k \in \mathbb{N}$ as input, decide whether there is an independent set of size $\geq k$.

search also given $k \in \mathbb{N}$ as input, find an independent set of size $\geq k$, or output that there is none.

opt find the size of a largest independent set.

search opt find a largest independent set.

These problems are all equivalent to each other under polytime Turing reductions, as we now show.

(search to decision) Given an oracle A solving decision maximum independent set and an input G, k, we can find an independent set of size $\geq k$ if there is one with the following algorithm

$B^A(G, k)$
  if not $A(G, k)$, return $\perp$
  if $k = 0$, return $\emptyset$
  let $x \in V(G)$
  $G_0 \leftarrow G - \{x\}$
  $G_1 \leftarrow G - \{x\} - N_G(x)$
  if $A(G_1, k-1)$, return $\{x\} \cup B^A(G_1, k-1)$
  return $B^A(G_0, k)$

The algorithm is described recursively for exposition, but it is a simple matter to remove the recursion and turn the above into an iteration.

(decision to search) Given an oracle B for seach, the reduction is $A^B(G, k) = (B(G, k) \neq \perp)$.

(opt to decision) Given an oracle A solving decision, we can find the size of a maximum independent set as follows.

$C^A(G)$
  for $k \leftarrow |V(G)|$ down to 0
    if $A(G, k)$, return k

(decision to opt) Given an oracle C solving opt, the reduction is $A^C(G, k) = (k \leq C(G, k))$.

(search opt to decision) Given an oracle A solving decision, we can find a maximum independent set by first finding the size of a maximum independent set and then searching for one of that size. I.e. we merely combine 2 of the above reductions.

(decision to search opt) Given an orcacle D for search opt, the reduction is $A^D(G, k) = (|D(G)| \geq k)$.

1.2 Generalization

The above equivalence proof seems to depend on the fact that an instance of maximum independent set can easily be reduced to a smaller instance by making a single decision about the solution, namely whether a single node is in or out of the independent set. This property is called *self reducibility* by complexity theorists or *self-similarity* by CLRS. (we will refrain from completely formalizing this concept here). Though it appears to be a crucial property to conclude the equivalence, it actually is not since we can easily generalize the original optimization problem to become a self reducible version by also including a partial assignment to the output as part of the input. In other words, change the original problem X to the following new problem pref$X$:

input (x,y)

output z

objective f(x,yz)

constraint C(x,yz)

To show that search polytime Turning reduces to decision, we will need one more hypothesis. Say that the constraint C is *fair* iff it is polytime decidable and there is a polynomial p s.t. $\forall x, y\ (C(x, y) \to |y| \leq p(|x|))$.

Lemma 1. *Given that f is computable in polytime and C is fair, the search version of* pref$X$ *polytime Turing reduces to the decision version of* pref$X$*.*

Proof. Wlog suppose that the input alphabet is $\Sigma = \{0, 1\}$. Given an oracle A for decision, we can solve search as follows:

```
B^A(x, y, k)
  if not A(x, y, k), return ⊥
  z ← ϵ
  while f(x, yz) < k
    if A(x, yz0, k), z ← z0
    else, z ← z1
  return z
```

Since C is fair, the while loop terminates after a polynomial number of steps. □

Of course this only shows that pref$X$ is self-reducible. What about X itself? If X happens to be NP-complete, then X is self-reducible as well, in the sense that search reduces to decision, since search X reduces to search pref$X$, which reduces to decision pref$X$, which reduces to decision X, since X is NP-hard. This is not the recommended route however, since many NP-complete problems have a much more direct reduction, as maximum independent set above had.

2 SAT $\in$ NP-complete

Recall that CircuitSAT is the problem of deciding whether a given circuit has an assignment to its inputs that causes it to evaluate to true. We claim that CircuitSAT is NP-complete. Evaluating a circuit can be done in polynomial time and the number of inputs is bounded by the size of the circuit, so CircuitSAT $\in$ NP. To show that CircuitSAT $\in$ NP-hard, we will rely on a fact shown earlier: P defined with respect to Turing machines is the same as P defined with respect to p-uniform circuits. (The infamous Cook tableau part of the proof is hidden inside this step.) So suppose $L = \operatorname{dom} R \in \mathrm{NP}$ where R is a fair relation and

p is a polynomial s.t. $\forall x, y\ (R(x,y) \to |y| = p(|x|))$. Since R is fair, there is a p-uniform circuit family deciding R and generated by a polytime Turing machine A. For a circuit c, let $\mathrm{hardcode}(c,x)$ be the circuit c with its first $|x|$ inputs hardcoded to x. Then $x \mapsto \mathrm{hardcode}(A(|x| + p(|x|)), x)$ reduces L to CircuitSAT in polynomial time.

Next we show that SAT $\in$ NP-complete. SAT $\in$ NP since a CNF formula is just a special case of a circuit. To show that SAT $\in$ NP-hard, we reduce from CircuitSAT. Let C be a circuit with variables $x_1, \ldots, x_n$. Construct a new variable y_i for each gate (internal node) i in C. For each gate i with inputs coming from nodes $v_1, \ldots, v_k$, let $z_1, \ldots, z_k$ be the variables representing those nodes (so each z_j will be either an original variable x_l or a new variable y_l). For each assignment a to $z_1, \ldots, z_k$ that makes i false, create a new clause $C_{i,a}$ that is false iff $z_1, \ldots, z_k$ are given the assignment a. Then create a single unit clause representing the output of C. The resulting formula is satisfiable iff C is satisfiable.