CSC 373 Lecture 20

Term test 2: Thursday or Friday?

Review:

IP vs LP; NP vs co-NP

Today

- Integer primality and factoring
- One more transformation: 3SAT transform to SUBSET-SUM. We can use this to show that 2 identical machine makespan is NP hard. (Text transforms 3SAT to 3-Dim Matching and then 3-Dim Matching is transformed to SUBSET-SUM.

SAT at the root of a tree of NP completeness and some history

- We will postpone establishing *SAT* (or *Circuit SAT* as in CLRS) as the root of a tree of *NP* completeness and just take that as a fact. *SAT* was the set that Cook (1971) first used and he then showed that other problems were also *NP* complete (such as *CLIQUE*). Cook also noted that "integer factoring" was in NP but not necessarily complete. Karp soon thereafter provided a list of ~20 natural problems which were also NP complete and that was followed by thousands more. The Garey and Johnson book is perhaps the most referenced book in CS.
- The concept of *P* as a model for "efficient computation" was already in work by Cobham and Edmonds. Levin (in the FSU) independently defined *NP* completenesss but his work was not known outdside of the FSU until about 1973.

NP vs *co-NP*

- *co-NP*. We say that a language *L* is in *co-NP* if its complement (the strings not in *L*) is in *NP*. (We "don't worry about" strings that do not encode input instances.) Note that *P = co-P* but the (again almost religious) belief is that *NP* is not equal to *co-NP*. For example, what certificate could you use so that I could verify that a formula *F* is not satisfiable, or that *G* does not have a clique of size (say) $k = |V|/2$?

The NP vs co-NP belief

- By definition, *L′* poly time transforms to L iff the the complement of *L′* poly time transforms to the complement of *L*.
- Also if *L′* transforms to *L* and *L* is in *NP*, then *L′* in *NP*. (Does not follow for poly time reduction.)
- It follows that *NP = co-NP* iff any *NP* complete set (with respect to transformation) *L* is in *co-NP*.
- So if *L* and its complement are both in *NP* we then have "strong evidence" that *L* is not *NP complete*.
- If P is not equal to NP, then it can be proven that there exists non complete L in NP - P

Primality and integer factoring

- Much of modern cryptography is based on the assumption of *NP* hardness (and "hard on average") and other assumptions. In particular some cryptography is based on the hardness of factoring integers. Note: here complexity is a function of the length of the integer input.
- There is a decision version of factoring; namely, we let *FACTOR = {(x,y): x* has a proper factor *z* which is *<=y*}. Clearly *FACTOR* is in *NP*. It is also clear that if *FACTOR* is poly time then we can factor integers in poly time. What is a little less clear is that *FACTOR* is in *co-NP*.

Primality and FACTOR in co-NP

- We need a little number theory to show that *PRIME = {x| x is prime}* is in *NP*. (It is obvious that *COMPLEMENT* is in *NP*. With some more number theory it was shown that *COMPOSITE* could be efficiently solved with randomization. Then in ~2000, two undergraduates and a faculty member at one of the IITs showed that *PRIME* in *P*. (It was known primality could be solved efficiently "in practice".)
- Just assumming *PRIME* in *NP*, we can show that *FACTOR* in *co-NP* using the prime decomposition of a integer. Hence although we believe factoring is difficult (on average) we believe it cannot be *NP* complete and thus *NP-P* contains non complete sets. We also believe then that a language *L* in *NP* and in *co-NP* is not necessarily in *P*.

What we need to show that PRIME is in NP

- We have the following fact: *p* is a prime iff Z_p^* (the multiplicative group of integers mod *p*) is a cyclic group
- This holds iff there exists a generator *g* such that $g^{p-1} = 1 \mod p$ and $g^{(p-1)/p_i}$ is not 1 for every prime divisor p_i of $p-1$.
- This allows for a recursive definition of *PRIME* as an *NP* set.
- FACTOR then is a good example of a problem in NP and also in co-NP but not believed to be in P.

Subset sum is NP Complete

- Let *U* be a set of positive integers. *SUBSET-SUM* = {*(U,t)*: there exists a subset *S* of *U* whose elements sum to *t*}.
- That *SUBSET-SUM* is in *NP* is easy to show
- To show (weak) hardness we transform *3SAT* to *SUBSET-SUM* . I use the transformation in section 34.5.5 of CLRS illustrated in fig. 34.19
- In fact, it is *NP* complete when we restrict *t* to be exactly half of the sum of elements in *S*. This is called the *PARTITION* problem. See old 364 notes

Figure 34.19 of CLRS

[redacted]								
v_1'	=	1	0	0	0	1	1	0
[redacted]								
v_2'	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
[redacted]								
s_1	=	0	0	0	1	0	0	0
s_1'	=	0	0	0	2	0	0	0
[redacted]								
s_2'	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
[redacted]								
s_4	=	0	0	0	0	0	0	1
s_4'	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

Figure 34.19 The reduction of 3-CNF-SAT to SUBSET-SUM. The formula in 3-CNF is $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, $C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$, and $C_4 = (x_1 \vee x_2 \vee x_3)$. A satisfying assignment of ϕ is $\langle x_1 = 0, x_2 = 0, x_3 = 1 \rangle$. The set S produced by the reduction consists of the base-10 numbers shown; reading from top to bottom, $S = \{1001001, 1000110, 100001, 101110, 10011, 11100, 1000, 2000, 100, 200, 10, 20, 1, 2\}$. The