CSC 373 Lecture 19

Review: NP sets, NP completeness, a tree of NP complete problems.

Reducing a search/optimization problem to a corresponding decision problem.

In tutorial, 3SAT transforms to directed HC

- IP vs LP
- NP vs co-NP
- Integer primality and factoring
- One more transformation: 3SAT transform to subset sum. Thus makespan is NP hard.

NP Sets (decision problems)

- What do these sets (say SAT and CLIQUE) have in common? They both can be easily "**verified**" by a succinct "**certificate**".
- For example, suppose I am "all powerful" (or perhaps just as good, suppose I am just a very lucky at guessing).
- Then if I want to prove that F is in SAT, I show you a satisfying truth assignment (call it *tau*) and then you (or an efficient algorithm) can easily verify that *F* is satisfied by *tau*. *Tau* is the succinct certificate.
- Similarly if I want to convince you that *(G,k)* is in CLIQUE, then I show you a subset of *k* nodes *V'* and you verify that *V'* is a clique in *G*.

The definition of an NP set

- Let *L* be a set (i.e. a subset of strings over some finite alphabet). Then *L* is in ***NP*** if there exists a polynomial time predicate (i.e. 0-1 valued function) *R(x,y) and polynomial q such that x in L* iff there exists a *y: |y| <= q(|x|)* and *R(x,y)* is true (i.e. *R(x,y) = 1*). That is, every x in L has a succint certificate y (where the poly q defines "succinct") that allows for efficiently verifying that *x* in *L (*where poly time *R* defines efficient verification) .

All the problems studied to date have corresponding NP decision problems

- (Job) Interval scheduling decision problem: For a set *S* of weighted intervals (resp. jobs for the JISP problem), and bound *W*, does there exist a subset of intervals (jobs) with profit at least *S*.
- The knapsack decision problem: For a set of items, size bound *W* and value bound *V*, does there exist a subset of items with total size at most *W* and value at least *V.*
- For sets in polynomial time (i.e. in ***P***) no certificate is needed. Clearly *P* is a subset of *NP*.

Reducing a search/optimization to a corresponding decision problem

- The Sat search problem is to find a satisfying assignment for F (if one exists) or say that *F* is not satisfiable. This efficiently reduces to the *SAT* decision problem: assuming that *F* is satisfiable, then choose any variable *x* and then use *SAT* to determine whether or not *F' = (F|x = true)* is satisfiable; if not then *F'' = (F|x = false)* is satisfiable. We then continue with *F'* or *F''*.
- To reduce the weighted JISP problem to the corresponding decision problem we first use binary search and the decision problem to determine the optimal value. Then we consider each interval (in the input set) and ask if we can remove that interval and still achieve the optimal value. If so we eliminate that interval and with the intervals that remain.

NP Complete Sets

- Let <= be a poly time reducibility (or poly time transformation). We will say that a set (decision problem) *L* is ***NP hard*** if for every *L'* in *NP* , *L'* <= *L*. Hence if *L* is *NP* hard but is also in *P*, then *P = NP*
- *L* is ***NP complete*** if *L* is in *NP* and *NP* hard. Hence *P = NP* iff there is any *NP* complete problem that is in *P*.
- Why do we religiously believe that *P* is not equal to *NP*? Because there are thousands of NP complete problems that have been thought about independently before and after the concept was defined and no one has been able to find a polynomial time algorithm for them. Moreover, the best algorithms for these natural *NP* problems are all exponential time (i.e. *c^n* for some *c* > 1)

The tree of NP completeness

- How do we show that a set *L* is *NP* complete? Usually (but not always) it is relatively easy to show that *L* is in *NP*. Usually it is the NP hardness that can sometime be quite non trivial to show. In fact, one might wonder how we show that any set *L* is *NP* hard since it requires showing something about every *L'* in *NP*. But suppose we do have one set *L* which is *NP* complete. Then if if we find another *L** in *NP* such that *L* <= *L** then *L** is also NP complete by the transitivity of <=. So starting with some *NP* complete *L* we can start to evolve a tree of *NP* complete problems.

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.

Integer programming *IP*

- *IP is an* example of a widely studied *NP* complete problem.
- It is relatively easy to establish that *IP* (and {0,1} IP) is *NP* hard.
- It takes a little work to prove that *IP* is in *NP*. WHY? This is unlike the usual situation where showing a problem is in *NP* is often quite obvious.
- We will later consider linear programming *LP* for which there does exist a (weakly) polynomial time algorithm (discovered in the 70s) and for which there has been a "practical algorithm" since the 40s

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.