CSC373S Lecture 19

- We now begin the topic of local search. In some sense we already began that topic in that the Ford Fulkerson FF algorithm for max flow can be viewed as a local search algorithm but it is rarely presented in this context. Lets start with what I would call the vanilla or basic local search paradigm:

  ```
  Compute an initial solution $S$
  While there exists a "better solution" $S'$ in a "local neighborhood" $N(S)$
      $S := S'$
  End While
  ```

  Assuming we can compute some initial feasible solution (e.g. for the Max Sat, Max Cut, or Max Independent Set MIS problems), this algorithm is conceptually simple to implement and will either not terminate or will terminate in a *locally optimally* solution and the standard questions to ask are:
  1) Does the algorithm terminate and if so how fast?
  2) Upon termination how good is a local optimum compared to a global optimum?

  The worst case ratio (over all local optimum) between a local optimum and a global optimum is called the *locality gap*. As such the approximation ratio is at least as good as the locality gap and could sometimes be better if (say) we never reach a worst case local optimum from the initial solution.

  When we are computing a search problem (or a problem where just getting a feasible solution is an issue), the paradigm can be used and now "better" can mean "closer to being feasible".

- There are many many variations on this vanilla local search paradigm and even for the vanilla paradigm we need to explain things. Lets mainly consider the case where we can (efficiently) obtain an initial feasible solution. We have the following issue:

  1. How to choose the initial solution? Often one takes a naive or trivial solution, or a random solution, or a solution computed by a simple or efficient method such as a greedy algorithm.

  2. How to define the local neighborhood $N(S)$ of a solution S? In problems where a solution is a subset (like MIS) or equivalently a vector $(x_1, ..., n_n)$ then a natural neighborhood (but not the only one) is a bounded Hamming neighborhood; i.e. $z \in N(x)$ iff $\{i : z_i \neq x_i\} \leq d$ with d small.

  3. What do we mean by "better". For an optimization problem we usually mean "better" with respect to the given objective function. Such local search algorithms have been called "oblivious local search". If instead we interpret "better" with respect to some related potential function, this is called non

oblivious local search. (We will see an example of non-oblivious local search later.)

Aside: The word "oblivious" seems like a poor use of terminology. If I were defining my own terminology, I would replace the terminology of greedy algorithms by *myopic algorithms* some of which are greedy (or stingy for a cost problem) where greedy/stingy then is an adjective implying a most opportunistic local decision in each iteration. Similarly, I might call oblivious local search to be greedy local search if it makes the best improvement (with the neighbourhood) to the given objective function. But lets not invent new terminology.

4. For either oblivious or non-oblivious local search do we settle for any better solution or for the best solution (i.e. greedy local search) in $N(S)$?
5. Usually if we start with a feasible solution we assume that $N(S)$ means only feasible solutions but that does not have to be the case.

Beyond all these issues, most heuristic (and one might say practical) applications of local search allow ways to escape local optima in some controlled way. Usually this is done in some randomized way and the terminology used is "stochastic local search". Perhaps the best known generic method in this regard is a parameterized method called "simulated annealing". For satisfiability problems, there is a popular class of stochastic local search methods under the name WALKSAT.

There are books written about stochastic local search and in general there is a common belief that local search style methods are often the best approach to solving optimization problems although the methods are often not analyzed (with regard to say locality gap or some sort of "expected locality gap" with the expectation over random inputs and/or randomization in the algorithm)). The use of randomization in local search methods may be another reason this topic is delayed in algorithm courses. But given the prominence of local search as a practical methodology, I think it should be introduced earlier in courses and not viewed as an advanced technique.

- Although, local search usually does not yield an optimal algorithm (i.e. the local optimum is not a global optimum), there are two very important settings where (oblivious) local search does produce optimal solutions, namely

  1. The simplex method for linear programming LP solving can be thought of as local search (assuming we can start with a feasible solution); however, from a worst case perspective all currently used "pivot rules" (for choosing a neighboring solution) can be shown to take exponential time to terminate. We will be discussing LP relaxations of integer programming IP in the last two weeks.
  2. The Ford Fulkerson based methods for max flow algorithms (including alternating path methods for matching). Here the neighbourhood $N(f)$ of a flow f is the set of all flows $f + f_\pi$ where f_π is the flow along an augmenting path π. Not that this neighbourhood can be exponential in size (i.e. there can be exponentially many

augmenting paths) but still we can search the neigbourhood in polynomial time as this is just a search for some $s-t$ path (or, in particular, a search for an augmenting path having the biggest residual capacity).

NOTE: I wish to emphasize again that local search is generally thought of as a heuristic and often is not analyzed; in many applications, some variant of local search often performs well "in practice". Trying to analytically understand why local search does so well in many applications is a subject of ongoing interest. But just like greedy algorithms or any conceptually simple approach, if nothing else having such algorithms gives us a benchmark for more sophisticated algorithms.

- We will briefly consider three applications of local search approximation algorithms where the neighborhood is essentially defined by a small Hamming distance. The first application is not in the text. We consider the unweighted maximum independent set (MIS) problem in $k+1$ claw free graphs. This problem directly relates to question 6 in problem set 3 which concerns the weighted version of this problem.

  For arbitrary graphs $G=(V,E)$, as we have mentioned before, the MIS problem is not only NP-hard, it is NP-hard to approximate within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ where $n = |V|$. But in many graph theoretic applications, we are often dealing with restricted classes of graphs. (We started the course discussing the interval selection problem which is just the MIS problem for interval graphs.) Now we will consider another restricted class of graphs for which we can approximate the MIS.

  A graph $G=(V,E)$ is a $k+1$ claw-free graph if the neighbourhood $Nbd(v)$ of every vertex v has at most k independent vertices. Why is this called $k+1$ claw-free? A $k+1$-claw in a graph $G=(V,E)$ consists of a vertex v (called the center of the claw) and $k+1$ adjacent vertices (called the talons of the claw). So saying that there are at mlst k independent vertices in the neighbourhood of any vertex is equivalent to saying that the graph does not have a $k+1$ claw.

  There are many graphs which turn out to be $k+1$ claw-free for small values of k. Many of these examples come from geometric intersection graphs. For example, the intersection graphs for unit length intervals (resp. unit squares, unit disks in Euclidean 2-space) are 3-claw free (resp. 4 claw free, 6-claw free). Note however, that arbitrary interval graphs are not in general $k+1$ claw-free for any fixed k. (There is a common generalization of $k+1$ claw-free graphs and interval graphs but we will not discuss that in this course.) Another interesting example of $k+1$ claw free graphs comes from the k-set packing problem where the input is a collection of k-sets (i.e. sets of cardinality at most k) and the goal is to find a maximum size subcollection of non intersecting sets. Set packing is the underlying combinatorial problem in combinatorial auctions.

  It is not difficult to see that a very natural greedy algorithm yields a k-approximation for the weighted (and hence unweighted) MIS problem restricted to the class of $k+1$ claw-free graphs. Namely, sort the vertices so that $w(v_1) \geq w(v_2) \ldots \geq w(v_n)$ and greedily add vertices to form a maximal (but not necessarily maximum) independent set (i.e. add a vertex u to the current independent set S iff $S \cup \{u\}$ is still an independent set.

Aside: 3 claw-free graphs are called claw-free and the weighted MIS problem can be solved optimally (in poly time) for 3 claw-free graphs and becomes NP-hard for $k \geq 4$ claw-free. The 3 claw-free result is not an easy result.

We wish to consider a very natural local search algorithm for the MIS problem on $k + 1$ claw-free graphs (in the hope of improving the k approximation bound). As stated in the problem set a basic oblivious local search for the weighted MIS problem (restricted to $k + 1$ claw-free graphs is the following:

```
$S := \emptyset$
While $\exists u \in V - S$ such that $w_u > w(N(u) \cap S)$
    $S := (S - N(u)) \cup \{u\}$
End While
```

For an unweighted graph this simplifies to :

```
$S := \emptyset$
While $\exists u \in V - S$ such that $S \cup \{u\}$ is an independent set
    $S := S \cup \{u\}$
End While
```