1 Summary of Material

This week, we are trying to solve NP-complete problems. We have a number of tools:

1. **Restrict the problem.**
   Do you really need to solve that NP-complete problem, or can you solve an easier one?

2. **Local search.**
   Local search is a heuristic that involves starting with a tentative solution for the problem and comparing it to solutions that are one step away. The procedure for local search is:

   - Define a cost function f. Suppose our goal is to minimize f.
   - Define a neighborhood function N.
   - Pick a starting point x.
   - While there is $y \in N(x)$ such that $f(y) < f(x)$, set $x = y$.
   - Return the solution.

   There are many variations of local search:
   Hill-climbing: The name for the basic method described above.
   Metropolis: Pick a random $y \in N(x)$. If $f(y) < f(x)$, set $x = y$. If $f(y) > f(x)$, set $x = y$ with some probability.
   Simulated annealing: Like Metropolis, but where the probability of moving to a higher-cost neighbor decreases over time.
   Tabu search: Like the previous two, but with memory in order to avoid getting stuck in suboptimal regions or in plateaus ("taboo" areas).
   Parallel search: Do more than one search, and occasionally replace versions that are doing poorly with copies of versions that are doing well.
   Genetic algorithm: Keep a population of searches that changes over time via "breeding" of the best-performing searches.

   The choice of neighborhood function N determines the features of the state space, so picking a "nice" neighborhood function is extremely important in designing local search algorithms.

3. **Approximation algorithms.**
   An approximation algorithm does not try to find the best solution, but one within some *approximation ratio* of the best solution. Some examples include:
   Vertex cover: Repeatedly choose an edge and throw both vertices into the cover. Approximates within a factor of 2.
   Max Cut: Hill climbing: Move a vertex across the cut if doing so will improve the number of crossing edges. Approximates within a factor of 2
   Euclidean traveling salesperson: Find an MST, walk from node to node. Approximates within a factor of 2.

4. **Randomness.**
   Often quick and with good bounds on the expected result. Some examples include:
   Max Cut: Flip a coin to determine which side of the cut each vertex is on. Approximates within a factor of 2.
   MaxSat: Relax a linear program to get probabilities for whether a variable should be true or false. Then do randomized rounding. Approximates within a factor of $1 - 1/e$.

However, the PCP Theorem also tells us that our ability to find approximations is limited – for example, one statement of the theorem shows the impossibility of getting a $(1-\epsilon)$-approximation for 3SAT.

2 Examples

2.1 Hopfield graph

A Hopfield network is an undirected graph. Each node v is assigned a state $s(v) \in \{-1, 1\}$. Every edge (i, j) has an integer weight w_{ij}.

For a given configuration of node state assignments, we call an edge (u, v) *good* iff $w(u,v)s(u)s(v) < 0$ and *bad* otherwise. A node u is satisfied if the sum of the weights of the good edges incident to u is larger than the sum of the weights of the bad edges incident to u, i.e.

$$\sum_{v:(u,v)\in E} w(u,v)s(u)s(v) \leq 0$$

A configuration is *stable* if all nodes are satisfied. Use local search to find a stable configuration.

Exercise. *Define a cost function f.*

Exercise. *Define a neighborhood function N.*

Exercise. *Explain why this algorithm will always succeed in finding a stable configuration. Remember that there are two parts to succeeding: you must terminate and must be correct.*

2.2 Approximation algorithm for disjoint paths

Given $G = (V, E)$ and sets of terminal vertices $S = \{s_1, \ldots, s_k\}$, $T = \{t_1, \ldots, t_k\}$, connect as many pairs (s_i, t_i) as possible with pairwise edge-disjoint paths.

This is NP-complete, and Kleinberg and Tardos tell us that polynomial-time approximation ratios much better than $O(\sqrt{|E|})$ do not exist. However, there is a simple greedy algorithm that achieves an approximation ratio of $2\sqrt{|E|}$. We call this the Greedy Disjoint Paths algorithm.

Exercise. *Give one or more greedy algorithms that you think solve this problem.*

Exercise. *Prove that the Greedy Disjoint Paths algorithm (as described by your TF) is a* $2\sqrt{|E|}$ *approximation algorithm.*