1

Main Idea: Keep a list of solutions you've tried so far.

Do steepest ascent hill-climbing:

move to the best neighbor

that you have not already

been to even if that manus

going downhill.

Problems:

small problem: can be slow to check if a solution has been seen before

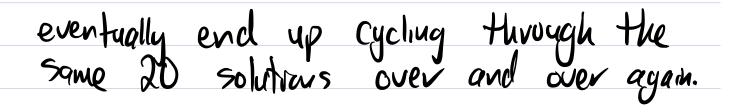
huge problem: His would use way too much memory

Fix (quick sketch):

#1) Keep a list only of the N most recent things seen.

#2) Don't even keep the whole solution, just keep the tweak that created it.

Fix #[When you see a solution you add it to the "tabu list" for some # of iterations (2), called the tabu tenure. In code: d=dict() key = solutions values = next time this solution is allowed to appear Keep track of which iteration you've on. When you go to a solution S at iteration N, you set d[S] = N + L Ex. If you're on it. 1,252 and L = 100 then you set d[S] = 1352. Whenever you want to go to a solution S. check d[5]. - If d[s] doesn't exist, you're good.
- If d[s] exists and is ≤ the current iteration #, you're also good. Problems? - Cycling behavior - if L=20, you rould



- Storing whole solutions still is not to tally ideal.

Fix #2: Keep the idea of tabu tenure, but on top of that, instead of remembering whole solutions, just remember the tweak you did.

Ex: Knapsack N=6

\$1,4,53 twent \$1,3,4,53 (add 3) Could do'

* don't remove 3 for 20 moves.

* could remove, but don't readd for 20 mouls

* or both.

Er: L=500 21,2,3,4,, 203 all combinations of 10 items being in or out $-2^{10} = 1024$

Benefit:
-Less to remember
few comparents to go in a cycle,
- Prevents repeatedly changing just a few components to go in a cycle, and forces more exploration.
Vague Pseudocode:
generation = 0
taboo = dict()
taboo_tenure = 20 (or whatever)
X=random element of search space
while True:
generation += 1
noighbors = nhhd(x) # each noighbor is
a pair (s, m) where s is the solution,
s is the solution,
and m is the
move that turns
x -> 5.
x, move = the pair (5,m) in
Theighbors such that m is
not a key in "taboo" or
[neighbors such that m is not a key in "taboo" or taboo[m] = generation] with the highest score
with the highest score

taboo[nove] = generation + taboo_tenure

Advanced Topics:

- * Sometimes using just the "move" as the taboo is too restrictive. In this case, you can try keeping the taboo list in terms of (move, score). So you only prevent a move if it would lead to the same score you had the last time you did the move.
- * Aspiration Criteria: You can decide ahead of time to ignove the taboo list under certain circumstances, e.g., if the new solution is the best you've ever seen.
- # If deighborhoods are two large:

 (i) change the tweak function, possibly allowing solutions that violate the constraints

Ex: Knapsack
old = zero or 1 items out and zero or 1 items in
zero or 1 items in
$O(n^2)$
new = add 1 item or remove 1 item
0(n)
might allow solutions over capacity,
so penalize them
(2) instead of generating the whole
(2) instead of generating the whole neighborhood, just generate K random tweaks and pick the
random twenks and pick the
best of those.