Wed, Apr 24, 2024

* HW 6 assigned
* Course Evals open

Topic 13 - Tabu Search

"Tabu" = "Taboo"

Think about H-C.

Walk up a hill and get stuck.
What do you do?

(1) Random restarts
(2) Sometimes go downhill according to a probability (SA)
(3) Go to the best location nearby that we haven't already tried even if it's downhill. (Tabu Search)

Like Steepest Asc. H-C but forces more exploration.

Main idea:

- ✫ Keep a list of solutions you've tried/visited.
- ✫ Do steepest-ascent H-C: move to the best neighbor that we have not previously visited, even if that means going downhill

✫ Only works for discrete problems [because we need a finite neighborhood]

Continuous Problem ⟶ Discrete Problem

Issues:

- Small issue: It might be slow to check if a solution has been seen before if we have a giant list of solutions.

Always use sets, not lists!

- Bigger issue: this would use up a lot of memory.

Fix #1:

When you visit a solution you add it to the "tabu list" for some # of iterations L.

tabu tenure

In code: `d = dict()`

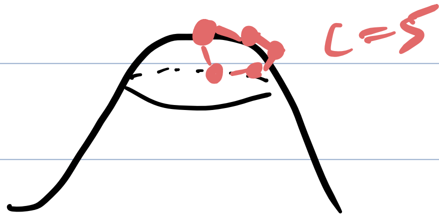
keys = solutions
values = first step where they can be revisited

* Keep track of what # step we're on as we go.
* When you see a solution s, at step #N, we assign $d[s] = N + L$.
* When you want to move to a new solution s', check $d[s']$. If it's $\leq$ current iteration, then this is allowed. If s' is not a key at all, also allowed. If $d[s'] >$ current iteration.

not allowed, move on to next best solution in the neighborhood.

Ocassionally prune the dictionary to remove entries that are now allowed.

This fix creates a new problem:
"Cycling" – if $L=20$, we might eventually just end up cycling through the same 20 solutions over and over again.



Fix #2: Instead of banning whole solutions once we see them, we're going to ban just the "moves" that created them.

Ex: Knapsack, $N=6$

$\{1,4,5\} \to \{1,3,4,5\}$ (add 3)

Some possibilities:

* don't remove 3 for the next L moves
* you can remove it, but don't add it back for the next L moves
* or some other creative idea

- This is less to remember.
- Prevents cycling by making it harder to just mess with a few items back and forth.

Pseudocode:

```
generation=0
taboo= dict()
taboo_time = 20   # depends on the problem
x = random elt. of search space
while True:
    generation += 1
    neighbors = nbhd(x)   # each neighbor is a
                          pair (s,m) where s is
                          the new sol (the neighbor)
                          and m is the "move"
                          that turned x->s
```

```
new_x, move = the pair (s,m) in neighbors
                with the highest score subject
              to the constraint that either
              m is not a key in taboo
              OR taboo[m] ≤ generation
x = new_x
taboo[move] = generation + taboo_time
```

TSP code