Algorithms Challenge 2 except I give up and throw heuristics at it

Tunan Shi

special thanks to Jakub Bachurski

2022

Results on validation data

Reviewed on Monday, 28 February 2022, 3:27 PM by Automatic grade
Grade: 77.4 / 100.0

Assessment report [-]

Running Python tests

```
Baseline score: 0.6565255021842563
Iter        0 (T 1.00):        Energy 14871.590328     Peak 14871.590328
Iter    30000 (T 0.81):        Energy 12358.243214     Peak 15018.589196
Iter    60000 (T 0.64):        Energy 11594.654817     Peak 15018.589196
Iter    90000 (T 0.49):        Energy 10977.072860     Peak 15018.589196
Iter   120000 (T 0.36):        Energy 10694.511374     Peak 15018.589196
Iter   150000 (T 0.25):        Energy 10268.601422     Peak 15018.589196
Iter   180000 (T 0.16):        Energy 10075.818102     Peak 15018.589196
Iter   210000 (T 0.09):        Energy 9966.013839      Peak 15018.589196
Iter   240000 (T 0.04):        Energy 9868.380569      Peak 15018.589196
Iter   270000 (T 0.01):        Energy 9809.415561      Peak 15018.589196
Iter   300000 (T 0.00):        Energy 9794.143209      Peak 15018.589196
Energy: 9794.14320924788 Score: 0.7737943053714852
FINAL SCORE: 0.7737943053714852
```

Figure: *Results can vary from 76.5-77.5*

How did I get here

- I wanted to code a baseline solution from standard techniques so that I had something to compare my future code against
- It turned out to beat like everything else I tried
- I'll take it?

Simulated annealing idea

- At the start, explore a bunch of different solutions across the solution space
- Go down one of the solution spaces that seem to be promising
- As you explore more and more solutions, start to pick off the obviously bad choices
- Towards the end, just try and optimise your current solution as much as possible

Lends many parallels to metal cooling in real life, and we borrow some of the terminology

Simulated annealing crash course

- How bad a solution is is called its energy
  - We want our solution to have low energy
- We make alternate solutions by modifying the solution such that the energy is only slightly affected
  - Example: just swap two of the elements in our solution
- Temperature: how willing you are to explore different solutions at the expense of transitioning to a higher energy state
  - Temperature decreases over time
  - When temperature reaches 0, we only allow solutions of lower energy

Basic algorithm

```
# How likely we are to accept a new solution with
# energy sp over an old solution with energy s
def prob(sp, s, T):
    if sp < s:
        return 1
    return np.exp(400*(s-sp)/(s*T))

# Run simulated annealing for some amount of iterations
def anneal(iters):
    sol = some starting solution # can be just randomness
    temperatures = np.linspace(1, 0, iters+1)[1:]
    for t in temperatures:
        new = sol but make a random swap
        if prob(e(new), e(sol), t) > random():
            sol = sol_new
    return sol
```

So... now what?

- Tune the algorithm!
  - Maybe it would work better if the temperature decreased exponentially?
  - Maybe run it for more iterations?
  - That magic number in the `prob` function may need a change
- Make each step run faster!
  - Vectorisation
  - Calculating the change in energy for each swap instead of always recalculating the entire thing
- Give the algorithm a headstart?
  - Maybe it won't have to try so hard at the start if you give it an okay solution to begin with?
  - Some algorithms are better than others for this

Slides download

`https://spdskatr.github.io/posts/anneal/`

If this is a potential employer, please ignore the part about

1561601101127958281549251050400861368019203391488