MET CS 767 - Project 1

Name: Ryan Kophs
Date: 20 September 2016

1 Problem Statement

Use simulated annealing to best approximate the optimal inputs, (x_1, x_2, x_3) such that $\forall x \in \{x_1, x_2, x_3\}(-10 \leq x \leq 10)$ and such that the following function, $f(x_1, x_2, x_3)$, yields the minimal output:

$$f(x_1, x_2, x_3) : x_1^2 + x_2^2 + x_3^2 \tag{1}$$

2 Approach

2.1 General Simulated Annealing

I chose to construct a general similated annealing approach that could be used in any arbitrary problem domain, The following pseudocode defines a single sweep of a single run of the algorithm:

```
**Require:** $s_{init}$, Neighbor, E, P
  $s \leftarrow s_{init}$
  $e \leftarrow \mathrm{E}(s)$
  $t \leftarrow 1.0$
  $t_{min} \leftarrow 0.0001$
  $\alpha \leftarrow 0.9$
  $k \leftarrow 1000$
  while $t \geq t_{min}$ do
    for all $0 \leq i < k$ do
      $s' \leftarrow \mathrm{Neighbor}(s)$
      $e' \leftarrow \mathrm{E}(s')$
      $p_{accept} \leftarrow \mathrm{P}(e, e', t)$
      $p_{rand} \leftarrow \mathrm{Random}(0, 1)$
      if $p_{accept} > p_{rand}$ then
        $s \leftarrow s'$
        $e \leftarrow e'$
      end if
    end for
    $t \leftarrow t * \alpha$
  end while
  return $(s, e, t)$
```

Such a sweep would be conducted multiple times using the same pseudorandom generator for a single run in order to provide a better stochastic minimum to the problem. Then, multiple runs (each with multiple sweeps) would be performed. Each run uses a pseudorandom generator with a different seed.

The inputs and variables for the above formula are defined as the following:

- s_{init}: Initial state within the problem domain. This is to be chosen at random.
- Neighbor(s): Function that accepts a state, s, as input and outputs a randomly chosen local neighbor, s', within the problem domain. It should randomly select a neighber that is considered local to s.
- E(s): Function that accepts a state, s, as input and outputs the state's energy, e, with regards to the problem space. The energy should the the output of the problem function f with s as the input.

- $\mathrm{P}(e, e', t)$ Function that accepts the energy, e, of current state, s, the potential energy, e', of new state, s', and the annealing temperature, t. It outputs an acceptable probability, p_{accept}, such that the new state, s', is accepted as the current state, s, if $p_{accept} > \mathrm{Random}(0, 1)$. This function should return $p_{accept} \leftarrow 1$ if $e' < e$. Otherwise, $p_{accept} \leftarrow \exp(e - e')/t$.
- t: The progressively cooling temperature of the annealing system. Starts at a temperature of 1 unit and eventually reduced to t_{min}. For each iteration, t is reduced in the following way: $t \leftarrow t * \alpha$.
- t_min: The temperature to which t should reach before the system exists
- α: The rate of temperature decrease of t for each annealing iteration.
- k: The number of steps for each temperature iteration. During each step, a new potential state, s', and potential energy, e', are generated and conditionally set to s and e, respectively given an acceptance probability p_{accept}.

2.2 Problem-Specific Simulated Annealing

It is the responsibility of the problem space to define $s_{init}, \mathrm{Neighbor}, \mathrm{E}, \mathrm{P}$ as defined above. For the problem at hand, a state s is defined as a tuple of (x_1, x_2, x_3) values. s_{init} is defined as randomly chosen:

$$s_{init} \leftarrow (\mathrm{Random}(-10, 10), \mathrm{Random}(-10, 10), \mathrm{Random}(-10, 10)) \tag{2}$$

Futhermore, Neighbor is defined as a function that takes state, $s = (x_1, x_2, x_3)$, as input and outputs a local neighbor, s', in which a randomly chosen x_i for $i \in \{1, 2, 3\}$ is updated with a random value, $\mathrm{Random}(-10, 10)$:

$$\mathrm{Neighbor}(s) : s' \leftarrow s; s'_{i \leftarrow \mathrm{Random}(1,2,3)} \leftarrow \mathrm{Random}(-10, 10) \tag{3}$$

Furthermore, E is defined as a function that takes a state, $s = (x_1, x_2, x_3)$, as input and outputs an energy, e, set to the result of problem function, f, as such:

$$\mathrm{E}(s = (x_1, x_2, x_3)) : e \leftarrow f(x_1, x_2, x_3) \tag{4}$$

Furthermore, P is an acceptance probability function that takes as input a current energy, e, a potential energy, e', and temperature, t. It then outputs a probability, p_{accept} such that:

$$\mathrm{P}(e, e', t) : p_{accept} \leftarrow (1.0 \textbf{ If } (e' < e) \textbf{ Otherwise } \exp((e - e')/t)) \tag{5}$$

3 Implementation and Project Layout

The entire solution is implemented in JavaScript with a HTML frontend. Specifically, the algorithm is broken up into two files:

- `js/SimulatedAnneal.js`: General simulated annealing algorithm that takes $s_{init}, \mathrm{Neighbor}, \mathrm{E}, \mathrm{P}$ as input.
- `js/MinSumOfSquares.js`: Problem specific simulated annealing algorithm that implements $s_{init}, \mathrm{Neighbor}, \mathrm{E}, \mathrm{P}$, and leverages `js/SimulatedAnneal.js` for the general simulated annealing heuristic.
- `js/PRG.js`: A custom pseudo random generator that accepts a seed tuple of integerts, (m, c). This was necessary because pseduorandom number generators vary from browser to browser.

In addition, the project includes a front end which allows a user to configure a seed, i, for the PRG, a number of sweeps, n, for a single run of the algorithm, as well as values: t, t_{min}, α, k, for the the simulated annealing algorithm sweep. The frontend outputs a list of the states, $s_i \leftarrow (x_1, x_2, x_3)$ and energies, e, for the t_{min} of each i. It also outputs the minimum e of all sweeps (the stochastic minimum of the problem).

The front end also outputs an animation of the problem space with each state, s_i for each annealing iteration of each sweep. The animation is a cube in which each dimention is different value, x_i for $i \in \{1, 2, 3\}$. Over time, states

s_i, should converge towards the middle of the box where $(x_1, x_2, x_3) = (0, 0, 0)$.

This functionality is implemented in the following files:

- `img/axis.png`: Represents each of the dimentions of the animation - (x_1, x_2, x_3).
- `js/Chart.js`: Generates the animation using HTML's canvas element.
- `style/normalize.css`: Standardize the css styles across all browsers (i.e. fonts, sizes, padding, etc ...)
- `style/style.css`: Styles specific to the project frontend in `index.html`.
- `index.html`: Frontend HTML code to run algorithm and resulting animation.

3.1 Execution

Set up a simple python server to serve up `index.html` and corresponding `js/*`, `style/*`, `img/*` resources.

```
$ cd /path/to/project/
$ python -m SimpleHTTPServer
```

This should display something like: "Serving HTTP on 0.0.0.0 port 8000 ...". At that point, go to `http://localhost:8000` to view the frontend. (Note that this is not required if viewing the frontend with Google Chrome. In such, just open index.html in the Chrome browser: e.g. `file:///<path-to-project>/index.html`)

4 Inputs and Solution

4.1 Inputs

The following inputs were used for a single run (multiple sweeps). Each are the default within the configuration on the frontend:

- t: 1.0
- t_{min}: 0.00001
- α: 0.95
- k: 100
- *seed*: 14759809013
- *sweeps*: 25

4.2 Solution

The resulting, best-effort, optimal solution state was: $(x_1, x_2, x_3) = (0.000875, -0.00284, -0.000318)$. The solution was generated over 225 iterations of annealing to reach a temperature of 0.000104. The resulting minimum value of f was: $min(f(x_1, x_2, x_3)) = 0.00000895$.

The previous was repeated for multiple runs of the following seeds: $\{14759809013, 743, 113819723, 1931117\}$. The respective results were as follows:

- $seed = 14759809013$: $(x_1, x_2, x_3) = (0.000875, -0.00284, -0.000318)$, $min(f(x_1, x_2, x_3)) = 0.00000895$
- $seed = 743$: $(x_1, x_2, x_3) = (0.0000925, -0.000734, 0.00476)$, $min(f(x_1, x_2, x_3)) = 0.0000232$
- $seed = 113819723$: $(x_1, x_2, x_3) = (-0.00683, -0.0133, -0.00835)$, $min(f(x_1, x_2, x_3)) = 0.000293$
- $seed = 14759809013$: $(x_1, x_2, x_3) = (0.00108, 0.00891, 0.0120)$, $min(f(x_1, x_2, x_3)) = 0.000224$

Thus, the minimal of all runs is: $(x_1, x_2, x_3) = (0.0000925, -0.000734, 0.00476)$, $min(f(x_1, x_2, x_3)) = 0.0000232$. These results may be reproduced by simply using the defaults on the frontend web application.

5 Resources

- `https://en.wikipedia.org/wiki/Simulated_annealing`