Simulated Annealing and its use in the Identikit Software

Erin Geaney
CSEP521
3/12/07

Simulated Annealing and its use in the Identikit Software

In an effort to learn more about the universe and its origin, radio astronomers must observe the behavior of galaxies and their interaction with one another. One of the difficulties in this study is that galaxies evolve over billions of years. For this reason, it is beneficial for astronomers to use astrophysics simulations to predict galactic evolution. It is difficult to simulate galaxies because of the complexity of these systems, as well as the enormity of the data sets involved. Dr. John Hibbard, from the National Radio Astronomy Observatory (NRAO), a partner of the University of Virginia Department of Astronomy, worked on a computer simulation program, Identikit, to model galaxies. My undergraduate thesis focused on improving the Identikit software by providing a method to automatically match the simulation data set to the observational data set using simulated annealing.

Identikit

Dr. Hibbard's project, Identikit, is a visualization tool to match simulations of interacting galaxies to observational data of these systems. The project works with simulation data in seven dimensions; x, y, and z, the velocities of x, y, and z (Vx, Vy, and Vz), and time. Identikit also uses observational data in three dimensions, x, y, and Vz since these are the dimensions that astronomers can observe from Earth (x and y planar position and velocity orthogonal to Earth using the Doppler effect). This simulator plots the observational and model data in four of these orthogonal planes, (x, y), (x, Vz), (y, Vz), and (x, z). There are various menu items and keyboard commands to rotate and scale the data until the observational data and the model is best matched in all four planes, which are displayed to the user.

Before my project, the Identikit user had to manually manipulate the parameters until it appeared that the user had the best fit. For my thesis project, I implemented the optimization algorithm to automatically change these parameters to produce an optimal fit.

Optimization Function

To quantify the “fit” of two data sets, I decided to use a method of least squares, where the goodness-of-fit is quantitatively determined by using the squares of the sums of the distances between the points in each direction. The fit of two data sets are determined quantitatively by summing the squares of the distances from each simulation point to the closest observational point in each dimension. For each point in the simulation data set, the algorithm will find the point closest to it and calculate the differences in the x, y, and vz directions, since these are the three dimensions in the observational data. These differences are squared then summed to determine the fit of a point. The quantitative matching value is the average of the sum of the squares from all

the points. The time to do this matching increases linearly, O(n), with respect to the number of points which I verified through experimentation.

Optimization Algorithm

The goal of my project was to determine the values of the variables, such as x, y, and z rotation, which produce an optimal solution. In the case of this project, the optimal solution is where the variables produce simulation data that most closely matches the observational data.

One approach to solving optimization problems is using simulated annealing. This name originated in the annealing process in thermodynamics. Annealing is a process used in the fabrication of objects constructed of metal or glass. It corrects a defect caused when metal or glass is shaped and become prone to fracture (McLaughlin). Simulated annealing uses the principles of annealing to produce an optimal solution to a problem.

The process of simulated annealing begins with a possible solution randomly selected as a starting point. Then a new solution, or configuration, is created at random. If this solution is better than the previous, this solution is kept. If it is worse, it is kept with a probability related to the Boltzmann distribution. This probability is $e^{(\textit{value of worse solution} - \textit{value of best solution})/T}$.

T is the "temperature" variable that decreases in time, similar to how a hot temperature will decrease in time (during annealing). Another way to think of T is how much worse a solution can be and still be considered. As time progresses, this value will decrease. In other words, a solution will have to be closer to the current best solution in order to be considered. Since T is decreasing, the probability of accepting a worse solution will decrease over time, as the solution becomes closer to optimum. (McLaughlin) It will run as long as T is larger than some minimum T value. If T is set to infinity and the algorithm is run forever, the optimum solution will result. The implementer of this algorithm can change the T and minimum T values to allow the function to run a less amount of time or longer depending on the needs.

Implementation

There were several steps involved in implementing this algorithm. I had to integrate the optimization function with simulated annealing and set up the data structures to store the necessary information. The following are the steps in the simulated annealing algorithm:

- The T variable is determined with a GetT function which returns two standard deviations of the distribution of 100 random solution's match value. Two standard deviations is the common value used for T in simulated annealing.

    - I tried larger numbers of random solutions on the same data sets and the results were very similar, so I decided that 100 random solutions would be the best for the amount of time it took to execute.
- The minimum T value is set to some proportion of the original T. I added a parameter so the user can decide how long the algorithm should run. The longer the algorithm runs, the better the solution.
- Until T is less than the minimum T, a "for loop" is executed. This loop is executed 360*log360 times. This loop is the essence of the algorithm.
    - A random parameter is selected. Originally, the parameters include the x, y, and z rotation.
    - After the parameter is chosen, the value is changed to a random value.
    - If the match value of the new simulation data is less than the current data, it becomes the new current data.
    - If not, it is kept as the new current with a probability related to the Boltzmann constant.
    - If it is better than the best value obtained so far, it is also the new best value.
    - After the for loop finishes execution, T is multiplied by 0.9 and if it is still greater than the minimum T, the while loop is executed again. At the end of the algorithm, the simulation datalist is set to the best datalist.

In the implementation, the current best solution and current solution have to be stored. I created a structure called a move that contained an axis name and an amount, such as 85 and x for 85º about the x-axis. In order to optimize the space needed to store the data structures, I created a stack of these moves instead of storing both the current particle positions and the optimal particle positions. Each move needed to be saved because the order of the moves determines the data positioning. Every time I accepted a new current datalist, I added the move to the stack. If a new solution became the new best solution, I cleared all of the moves from the stack. At the end, I undid the moves in reverse order, the most recent first, to obtain the best solution.

From the debugging process and knowledge of simulated annealing, I discovered that the longer the algorithm runs, the better the fit. Therefore, I decided to add a parameter, called length, to the annealing function. This parameter, a floating-point number between 1 and 10, determines how long the algorithm will run. The minimum T value is determined from this length.

The following is pseudo code for the simulated annealing algorithm:

```
Annealing(datalist observationData, datalist simulationData, length)
      set BestMoves = empty   // will be a stack of moves to get back
                              // to the current optimal configuration

      set t = GetT(observationData, simulationData)
      set mint = (0.99 - ((length-1) / 10)) *t
```

```
set thismatch = match(observationData, simulationData)
set currentbestmatch = thismatch;
set currentmatch = thismatch;

while ( t > mint)
      for (i = 0; i < n*log(n); ++i)
            set parameter = random integer from 0 to 2
            set degrees = random number from 0 to 360

            switch (parameter)
                  case 0:
                        rotateanneal( degrees, 'x',
                        simulationData)
                  case 1:
                        rotateanneal( degrees, 'y',
                        simulationData)
                  case 2:
                        rotateanneal( degrees, 'z',
                        simulationData)

            thismatch = match(observationData, simulationData)

            // if new match is better than the current match,
            // set the current match to this match
            if (thismatch <= currentmatch)
                  currentmatch = thismatch

                  // if this match is better than the best match,
                  // set the best match to this match and reset
                  // the best move list
                  if (thismatch <= currentbestmatch)
                        empty the BestMoves list
                        currentbestmatch = thismatch


                  // if this match isn't better than the best
                  // match, add the move to the list of moves
                  // since the best match
                  else
                        Add this move to BestMoves


            // if this match isn't better than the current match,
            // accept with probability of Boltzmann constant
            else
                  set delta = currentmatch - thismatch
                  set boltzman = e^(delta/t)
                  randomprob = random number from 0 to 1

                  if (randomprob < boltzman)
                        Add this move to BestMoves

                  else
                        set the data positions to the previous
                        positions
```

```
      t = t*0.9

      PutBackBestConfiguratino(simulatedData, BestMoves)


// GetT takes two datalists as parameters, finds 100
// random solutions, and returns two standard
// deviations of the distribution of solutions
GetT(datalist observationData, datalist simulatedData)
      set tlist = empty // stores a list of moves
      set values = empty // stores list of random values from 0-360
      set sum = 0
      // find 100 random solutions
      for (i = 0; i < 100; ++i)
            for (j = 0; j < 3; ++j)
                  values[j] = random number from 0 to 360
                  rotateanneal( values[0], 'x', simulatedData)
                  add move('x', values[0]) to tlist
                  rotateanneal( values[0], 'y', simulatedData)
                  add move('y', values[0]) to tlist
                  rotateanneal( values[0], 'z', simulatedData)
                  add move('z', values[0]) to tlist
            results[i] = match(observationData, simulatedData)
            sum = sum + results[i];

      set mean = sum / 100

      // calculate variance
      set variance = 0
      for (i = 0; i < 100; ++i)
            variance += (results[i] - mean)*(results[i] - mean)

      // calculate standard deviation
      set standarddeviation = sqrt(var / (num - 1))

      return (2*standarddeviation)


rotateanneal(angle, axis, datalist s)
      set radians=angle/deg_per_rad
      foreach point in s
            set X = point.x
            set Y = point.y
            set Z = pointz
            set VX = point.vx
            set VY = point.vy
            set VZ = point.vz

            if(axis=='x')
                  point.y=Y*cos(radians)+Z*sin(radians)
                  point.z=-Y*sin(radians)+Z*cos(radians)
                  point.vy=VY*cos(radians)+VZ*sin(radians)
                  point.vz=-VY*sin(radians)+VZ*cos(radians)
            else if(axis=='y')
                  point.z=Z*cos(radians)+X*sin(radians)
                  point.x=-Z*sin(radians)+X*cos(radians)
```

```
                  point.vz=VZ*cos(radians)+VX*sin(radians)
                  point.vx=-VZ*sin(radians)+VX*cos(radians)
            else if (axis=='z')
                  point.x=X*cos(radians)+Y*sin(radians)
                  point.y=-X*sin(radians)+Y*cos(radians)
                  point.vx=VX*cos(radians)+VY*sin(radians)
                  point.vy=-VX*sin(radians)+VY*cos(radians)

PutBackBestConfiguration(datalist a, movelist moves)
      for each move in moves
            rotateanneal(-move.angle, move.axis, a)


match(datalist simulatedData, datalist observationData)
      set sum = 0
      foreach particle p in observationData
            sum = sum + minrms(p, observationData)
      return sum/(number of particles in observationData)

minrms(particle p, datalist a)
      set min = rms(first particle in a, p)
      foreach particle q in a
            if (rms(q, p) < min)
                  min = rms(q, p)
      return min

rms(struct particle a, particle b)
      return (a.x - b.x)^2 + (a.y - b.y)^2 + (a.vz - b.vz)^2
```

Integration into Identikit

I integrated the optimization algorithm into the previous Identikit software so the user can now press a button on the GUI, called Find Best Fit, and a dialog box appears. The user enters a floating point number between one and ten to tell the simulated annealing algorithm how long to run. This way, if the user does not have much time, they can get an estimated best fit in a few minutes. If time is not as much a constraint, such as letting the program run overnight, or the user has a faster computer, he can allow the algorithm to run for a longer time and produce a better fit. Figure 1 shows the new GUI before the user pushes the Find Best Fit button.

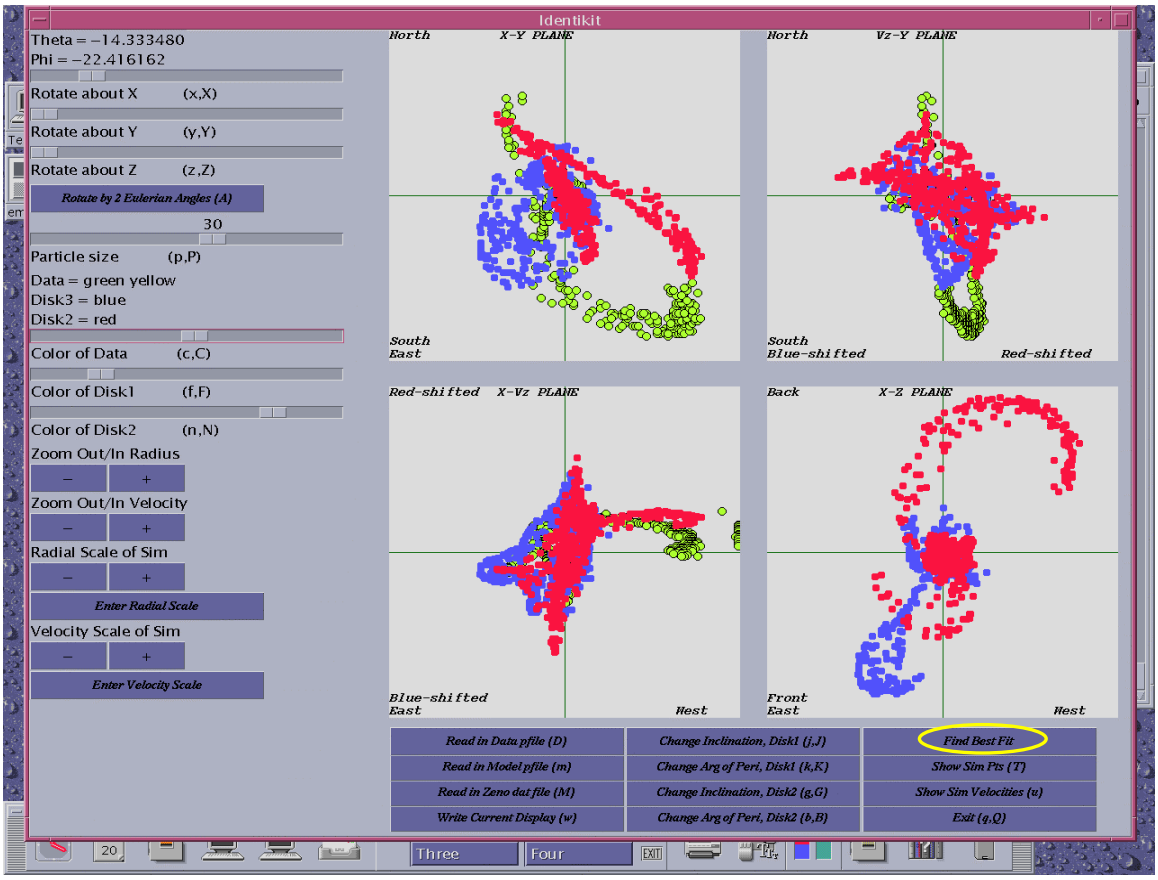

Figure 1. New Identikit GUI, before Find Best Fit is Pressed

On the right, the four planes, (x, y), (x, z), (Vz, Y), and (x, Vz), are displayed. In these planes, the green points are the observational data and the blue and red points are the simulation data. The goal of the user is to best match these data sets to each other. He can do this my manually modifying the parameters on the left or he can push the Find Best Fit button. Figure 2 shows the results after the user presses the Find Best Fit button and the algorithm has executed.

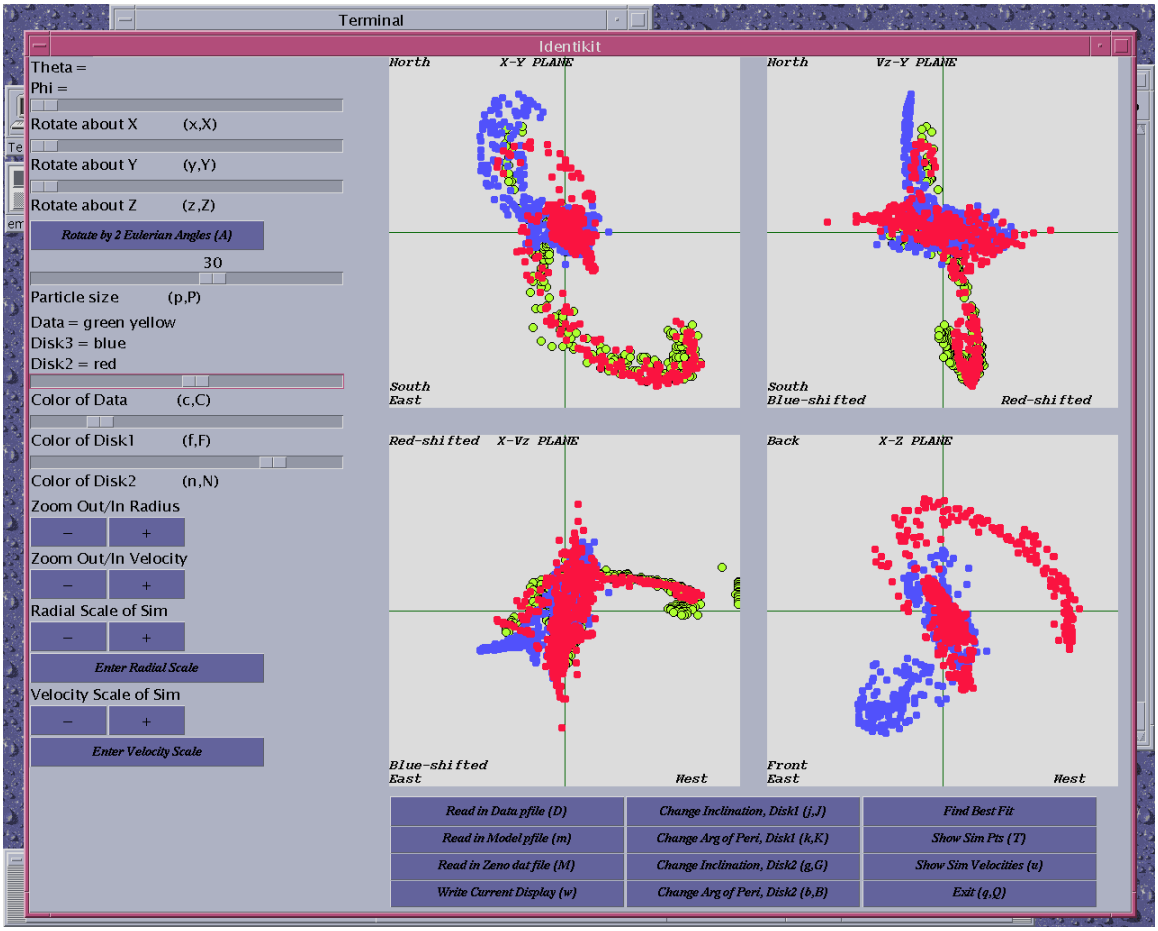

Figure 2. Screen Shot of Identikit after Find Best Fit is Executed

The figures show that the simulated annealing algorithm automatically produced a good fit between the observational and simulation data. By using an optimization function which averages how far each observational data point is from its closest simulation data point and implementing a simulated annealing algorithm determined the best combination of "moves", the user is able to automatically determine a good fit.

Bibliography

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. Cambridge, MA: The MIT Press, 1998.

Geaney, Erin. “Identikit: A Visualization Tool to Match Galaxy Models to Observational Data”. University of Virginia Fourth-Year Thesis, 2001.

Hibbard, John. “Identikit: An N-body/Data Visualization Tool.” http://www.cv.nrao.edu/~jhibbard/Identikit/. Available Oct. 24, 2000.

McLaughlin, Michael P. "Simulated Annealing." ACM Digital Library Sep 1989: Article No. 2.