7.2 Simulated Annealing

Ming Lu

2022-04-02

Simulated annealing (SA) falls into the category of stochastic thresholdmethods. This means that the algorithm starts with a randomly generated instance of a solution, and in successive iterations a neighbor of a current solution is randomly generated and their costs difference is compared to a given threshold. For simulated annealing the threshold at each iteration is either 0 (so that solutions that optimize the cost function F are accepted), or a positive random number that follows a probability distribution skewed towards solutions corresponding to smaller increases in cost, and depending on the difference of costs deltaF, and a global parameter T. In practice, these random thresholds control the frequency of making escape-moves in the following way. The parameter T is a non-increasing function which is slowly decremented at each time step, and this produces a reduction in the probability of accepting a worse solution, as this probability is governed by the negative exponential distribution of , with coefficient . The process is repeated until some stopping criteria holds, that is usually a combination of reaching a predefined number of iterations, or making no further moves (for better or worse solutions) for some number of steps. The general scheme of simulated annealing just described, for solving an optimization problem , is shown in Algorithm 7.2.

# 7.2.1 The Basics of Simulated Annealing

Simulated annealing was first proposed as an optimization heuristic in Computer Sciences by Kirkpatrick et al. (1983) for the design of VLSI circuits, and independently by Černý (1985) for a solution to the Traveling Salesman Problem. The idea for this heuristic comes from an analogy with the work of Metropolis et al. (1953) in statistical thermodynamics, who gave a method for simulating the energy changes of a solid in a heat bath from melting point down to a state of thermal equilibrium (i.e. cool state), by means of a cooling, or annealing, process. According to the laws of thermodynamics, the probability of having an energy increment at a certain temperature can be estimated by the equation

where is the Boltzmann constant. The Metropolis algorithm is a Monte Carlo simulation that successively makes random perturbations to the system, and computes the change in energy when passing from one state to the next. If there is a drop of energy (i.e.  ) then the new state is accepted, otherwise the new state is accepted with a probability given by Eq. (7.3). The process is repeated several times through a series of temperature decrements, which will lower the energy of the system until thermal equilibrium is reached.

The analogy of simulated annealing in combinatorial optimization to the thermodynamical annealing process of Metropolis should be now clear: think of instances of solutions as states of the physical system; cost function as energy; control parameter as the temperature; and an optimal solution as a cool (or low energy) state. The eventual convergence of the Metropolis algorithm to a state of thermal equilibrium is characterized in terms of the Boltzmann distribution, which relates the probability of the system of being at a certain state with a certain energy to the current temperature. From this fact can be derived an analysis of the convergence of simulated annealing, which we shall not do here and rather refer the reader to Aarts and Lenstra (2003, Chap. 4) for all the details of this analysis (see also Laarhoven and Aarts (1987)). A more important issue for practical purposes is the setting up of the parameters that control the simulated annealing heuristic. There are basically three parameters to consider: the stopping criterion, the cooling schedule and the neighborhood range. All three parameters are interdependent and the determination of their values depend on the nature of the optimization problem. However, observe that in general, from Eq. (7.3) setting , and solving for the temperature , we get

This leads to the following simple idea for determining an initial value for the temperature and a cooling schedule: For an arbitrary large number of steps, compute solutions at random in the neighborhood of initial solution and determine the average increase of the objective function over this set of solutions. Fix a high probability or and set initial temperature to . Then apply simulated annealing algorithm with initial temperature as and successive decrements given by the scheme , for , and a constant positive number less than but closer to 1 (the cooling parameter). A standard value for is . In applications of simulated annealing this static geometric cooling schedule is usually adopted for setting up the values for the parameter . The stopping criterion can now be taken to be the moment the temperature reaches a very low value . However, more often, and simpler, is to repeat the search for a large (fixed) number of iterations. A typical default value for is 10,000 . Finally, the neighborhood range can be adjusted beginning with a large value, so as to explore a large set of solutions, but successively reduce the radio for allowing escape-moves. We put this heuristic to work for estimation of the parameters of a process, in the next section.

# 7.2.2 Estimating a GARCH(1, 1) with Simulated Annealing

We solve the optimization problem proposed in Example 7.1, that is, the estimation of parameters for a GARCH(1,1) process, with the SA method. Recall that this is a maximization problem and therefore we must adapt the general SA heuristic (Algorithm 7.2) stated for a minimization problem. This is attained simply by turning

to , and tests of negativity to . The objective function is the likelihood estimator of a tuple , and given by Eq. (7.2). Algorithm presents the SA solution to this GARCH estimation. One can set the initial temperature , and the cooling parameter , to values calculated with the heuristics explained in the previous section. The stopping criterion is determined by reaching a maximum number of iterations. At each iteration, for a current solution , a new candidate solution is randomly generated by selecting one element and defining a perturbation , where is a i.i.d random number and is the neighborhood range; for .

We can program Algorithm in R, or an equivalent algorithm without much programming effort, using already existing R-packages for general purposes optimization.