MCMC Simulated Annealing Exercises.

Anatoli Iambartsev
IME-USP

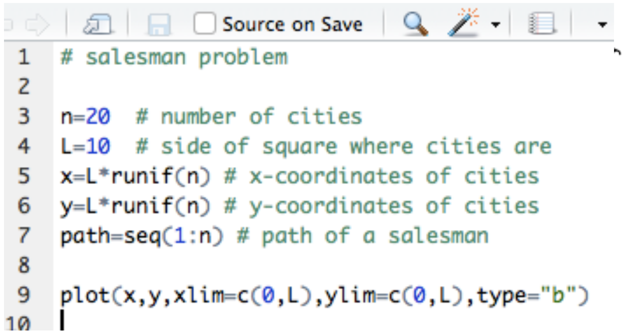
[Wiki] Salesman problem.

The *travelling salesman problem* (TSP), or, in recent years, the *travelling salesperson problem*, asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

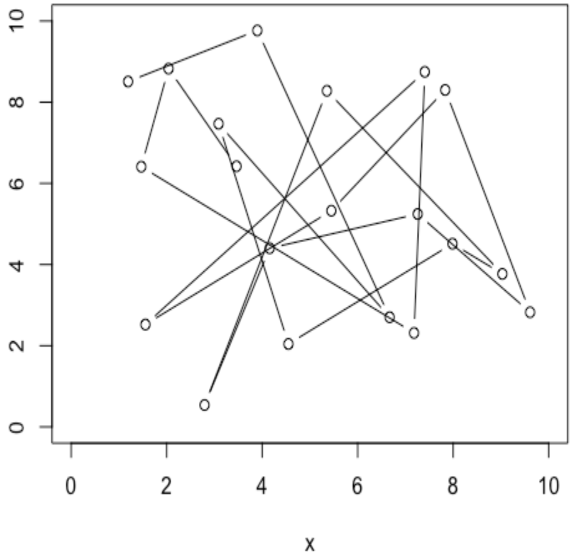
(NP-hardness (non-deterministic polynomial-time hard), in computational complexity theory, is the defining property of a class of problems that are, informally, "at least as hard as the hardest problems in NP". ... A common misconception is that the NP in "NP-hard" stands for "non-polynomial" when in fact it stands for "**N**on-deterministic **P**olynomial acceptable problems". Although it is suspected that there are no polynomial-time algorithms for NP-hard problems, this has not been proven. Moreover, the class P in which all problems can be solved in polynomial time, is contained in the NP class.)

Salesman problem.

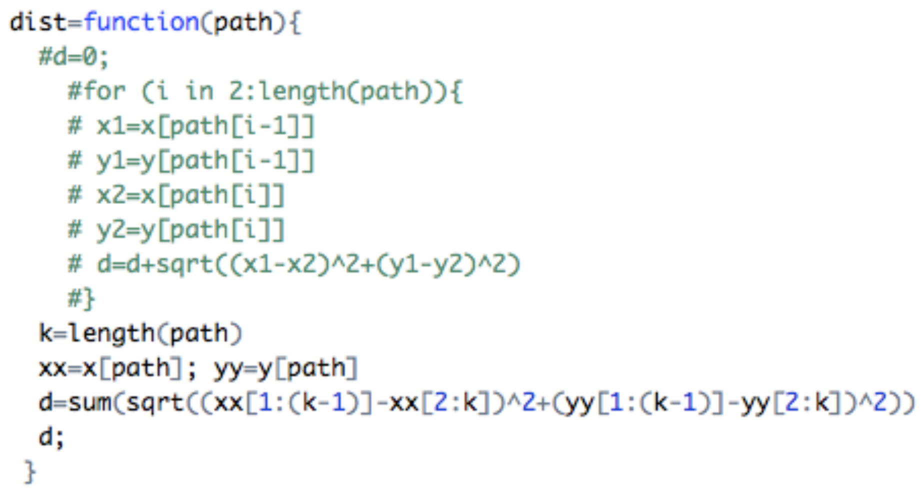
We will consider this problem without the condition "returns to the origin city".



```
# salesman problem

n=20  # number of cities
L=10  # side of square where cities are
x=L*runif(n) # x-coordinates of cities
y=L*runif(n) # y-coordinates of cities
path=seq(1:n) # path of a salesman

plot(x,y,xlim=c(0,L),ylim=c(0,L),type="b")
```



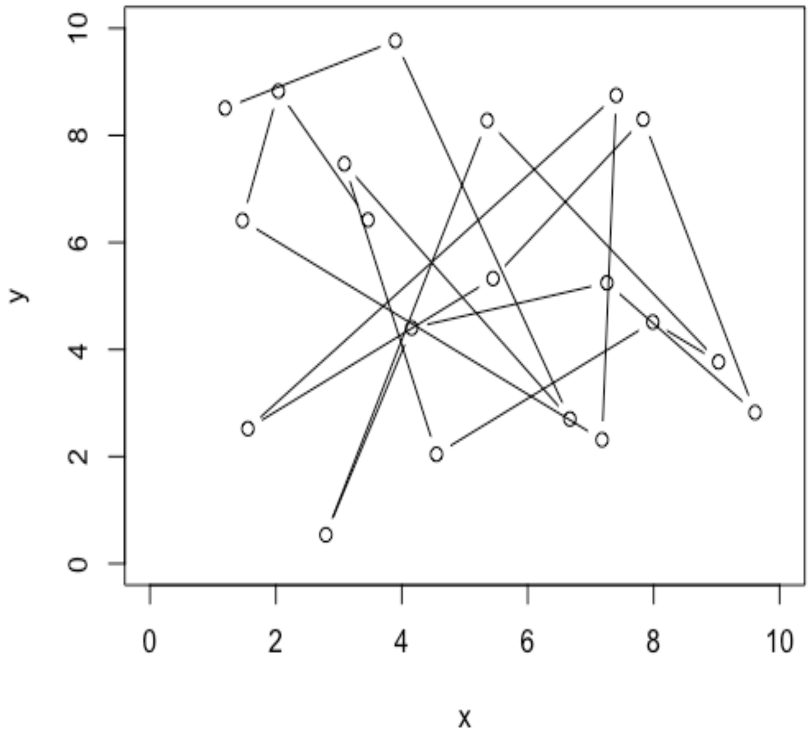
Salesman problem.



```
dist=function(path){
  #d=0;
    #for (i in 2:length(path)){
    # x1=x[path[i-1]]
    # y1=y[path[i-1]]
    # x2=x[path[i]]
    # y2=y[path[i]]
    # d=d+sqrt((x1-x2)^2+(y1-y2)^2)
    #}
  k=length(path)
  xx=x[path]; yy=y[path]
  d=sum(sqrt((xx[1:(k-1)]-xx[2:k])^2+(yy[1:(k-1)]-yy[2:k])^2))
  d;
 }
```

Salesman problem.

```
>
> dist(path)
[1] 93.98942
>
```



Salesman problem. Walks on the set of paths.

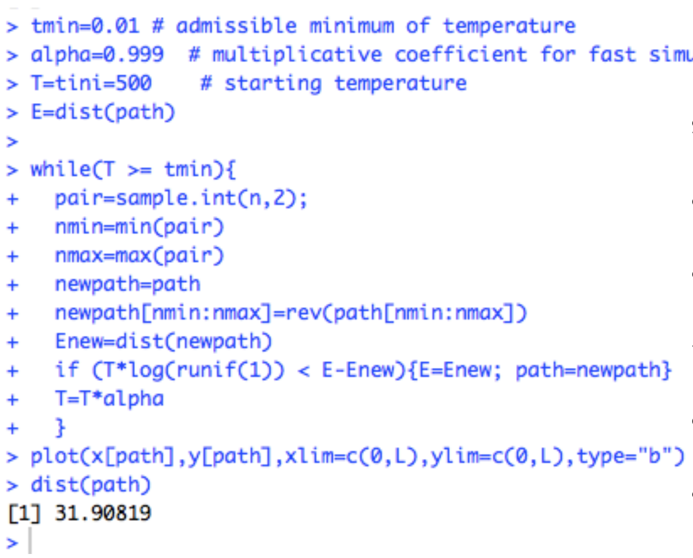
Choose a subpath of a path and revert the order

```
pair=sample.int(n,2);

nmin=min(pair)

nmax=max(pair)

newpath=path

newpath[nmin:nmax]=rev(path[nmin:nmax])
```

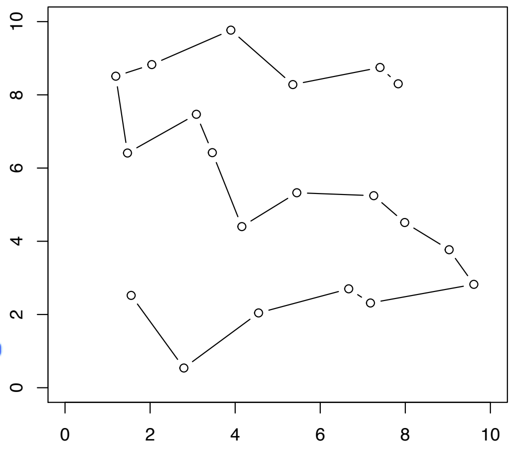
Observe that

$$\mathbb{P}(\texttt{path} \to \texttt{newpath}) = \mathbb{P}(\texttt{newpath} \to \texttt{path}) = \frac{1}{\binom{n}{2}}$$

Salesman problem.



```
> tmin=0.01 # admissible minimum of temperature
> alpha=0.999  # multiplicative coefficient for fast simulated annealing T[i+1]=alpha*T[i]
> T=tini=500    # starting temperature
> E=dist(path)
>
> while(T >= tmin){
+   pair=sample.int(n,2);
+   nmin=min(pair)
+   nmax=max(pair)
+   newpath=path
+   newpath[nmin:nmax]=rev(path[nmin:nmax])
+   Enew=dist(newpath)
+   if (T*log(runif(1)) < E-Enew){E=Enew; path=newpath}
+   T=T*alpha
+   }
> plot(x[path],y[path],xlim=c(0,L),ylim=c(0,L),type="b")
> dist(path)
[1] 31.90819
> 
```

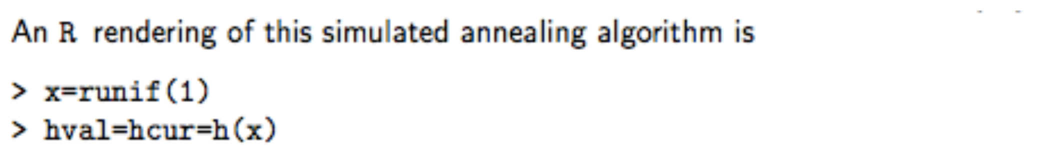


Salesman problem. Exercises.

1. Rewrite the algorithm for the salesman problem with the condition “returns to the origin city”.
2. Suggest another walk g on symmetric group S_n (set of all permutations of sequence $1, 2, \ldots, n$).
3. Instead of the (super-)fast annealing cooling schedule $T_{t+1}/T_t = \alpha \in (0, 1)$, use in simulated annealing different cooling schedule: $T_t = 1/\log(1 + t)$ (Boltzman’s annealing), $T_t = 1/t$ (Cauchy’s annealing).
4. Matching problem. In a square there are n red and n blue points. We connect one red and one blue point by an interval, the weight of this connection (matching) is the distance between them. The aim is to match any red point with blue one such that a total weight of matching should be minimal.

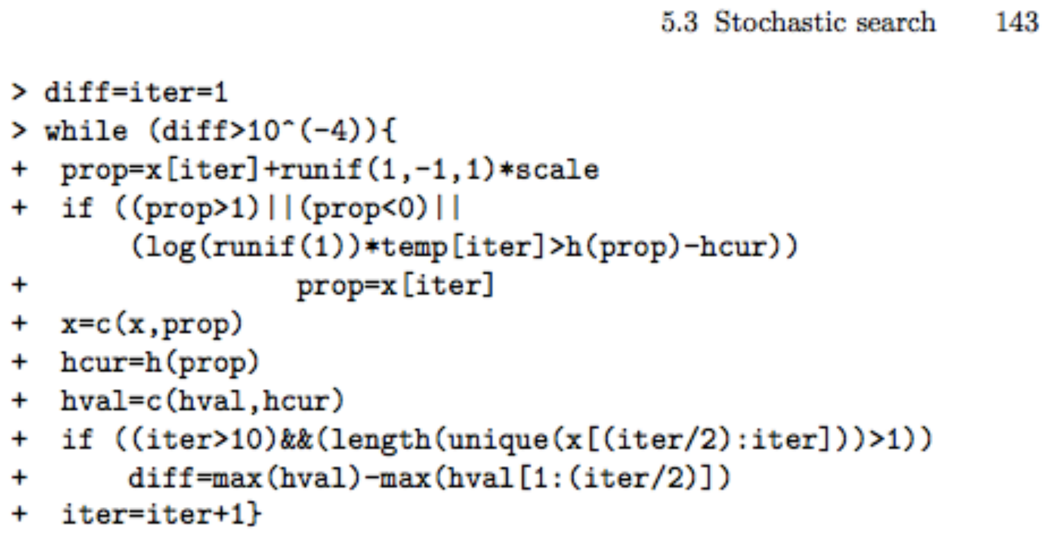
[RC] Example 5.8.

For the simple function $h(x) = [\cos(50x) + \sin(20x)]^2$, we can compare the impact of using different temperature schedules on the performance of the simulated annealing sequences. Note that, besides setting a temperature sequence, we also need to set a scale value (or sequence) for the distribution g of the perturbations as well as a stopping rule. Since the domain is $[0, 1]$, we use a uniform $U(-\rho, \rho)$ distribution for g and our stopping rule is that the algorithm will stop when the observed maximum of h has not changed in the second half of the sequence $\{x_t\}$.



An R rendering of this simulated annealing algorithm is

```
> x=runif(1)
> hval=hcur=h(x)
```



```
> diff=iter=1
> while (diff>10^(-4)){
+  prop=x[iter]+runif(1,-1,1)*scale
+  if ((prop>1)||(prop<0)||
       (log(runif(1))*temp[iter]>h(prop)-hcur))
+                prop=x[iter]
+  x=c(x,prop)
+  hcur=h(prop)
+  hval=c(hval,hcur)
+  if ((iter>10)&&(length(unique(x[(iter/2):iter]))>1))
+      diff=max(hval)-max(hval[1:(iter/2)])
+  iter=iter+1}
```

[RC] Example 5.8.

“The constraint involving unique is cancelling the stopping rule when no perturbation has been accepted in the second half of the iterations, meaning that the scale may then be inappropriate. (Note that the updates of `temp` and `scale` need to be included in the loop.)

For a scale defined by $\sqrt{T_t}$ and a temperature decrease in $1/\log(1+t)$, the sequence almost always ends up at a value close to the true maximum. Similarly, a scale defined by $5\sqrt{T_t}$ and a temperature decrease in $1/(1+t)^2$ leads almost certainly to the global maximum, as shown on Figure 5.8 (where the last example was obtained after several runs). Decreasing the scale by a factor of ten has a clear and negative impact on the performance of the algorithm.”

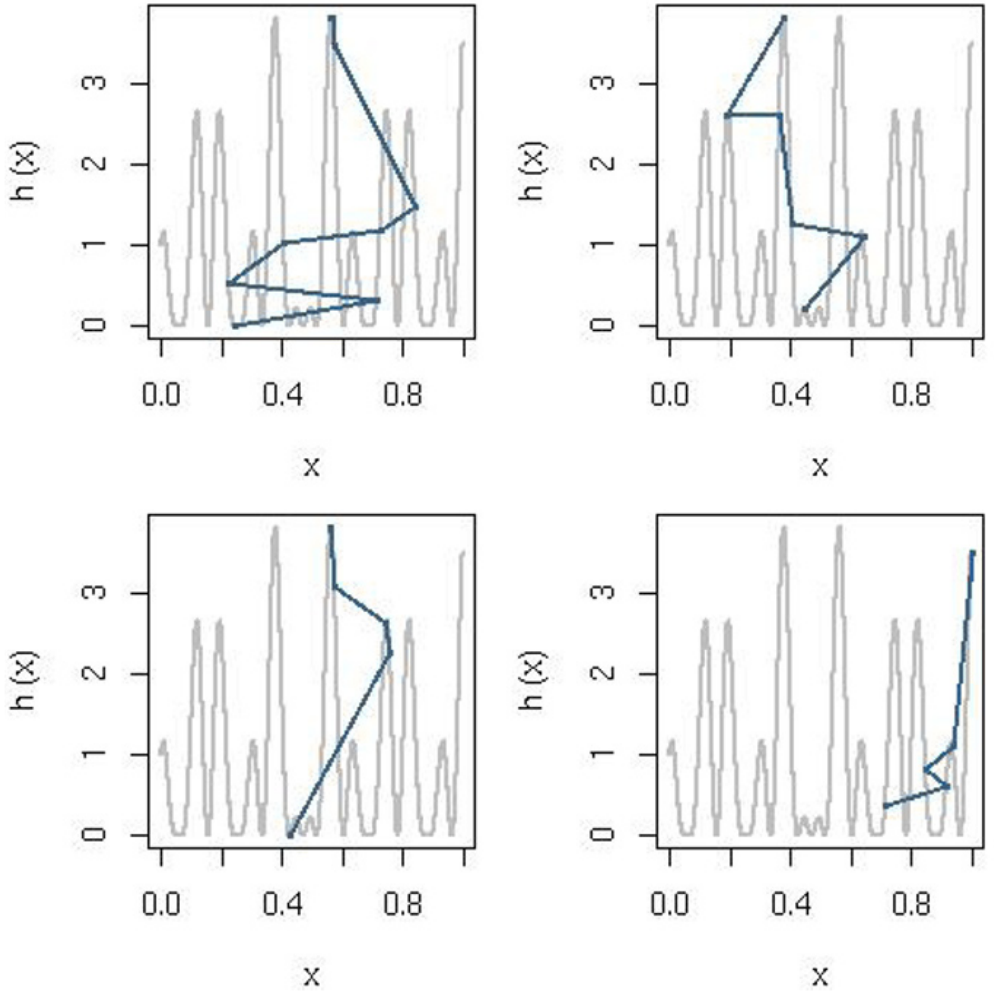


Fig. 5.8. Realizations of four simulated annealing sequences for $T_t = 1/(t+1)^2$ and $\rho = 5\sqrt{T_t}$ over the graph of the function h *(grey)*. Note that the points represented on the graph of h correspond to successive accepted values in Algorithm 2 and do not reflect the number of iterations.

[Wiki] Cooling schedule.

The physical analogy that is used to justify SA assumes that the cooling rate is low enough for the probability distribution of the current state to be near thermodynamic equilibrium at all times. Unfortunately, the relaxation time – the time one must wait for the equilibrium to be restored after a change in temperature – strongly depends on the "topography" of the energy function and on the current temperature. In the SA algorithm, the relaxation time also depends on the candidate generator, in a very complicated way. Note that all these parameters are usually provided as black box functions to the SA algorithm. Therefore, the ideal cooling rate cannot be determined beforehand, and should be empirically adjusted for each problem. *Adaptive simulated annealing* algorithms address this problem by connecting the cooling schedule to the search progress.

References.

[RC] Cristian P. Robert and George Casella. *Introducing Monte Carlo Methods with R*. Series "Use R!". Springer

Further reading.

Adaptive simulated annealing. `https://www.ingber.com/ASA-README.html`

Constantino Tsallis and Daniel A. Stariolo. *Generalized simulated annealing*, Physica A 233 (1996) 395–406.

Heikki Haario and Eero Saksman. *Simulated Annealing Process in General State Space*, Advances in Applied Probability, Vol. 23, No. 4 (Dec., 1991), pp. 866–893