Stochastic Optimization

conventional optimization problems can be solved by standard methods

e.g. *linear* or *quadratic programming* problems of the type

minimize $f(\vec{x})$ *subject to* $\{g_i(\vec{x}) = 0\}_{i=1}^{k}$, $\{h_j(\vec{x}) \geq 0\}_{j=1}^{m}$ *for* $\vec{x} \in \mathbb{R}^n$

by exact or numerical methods, e.g. gradient based search

difficult, hard, complex optimization problems: (one or several of the features)

– highly non-linear functions f

– f with many local minima

– many variables, i.e. high-dimensional vectors $\vec{x}$

– discrete variables, e.g. $\vec{x} \in \{0, 1\}^n$

standard methods (gradient based, etc.) often fail,
search for and identification of global minima is very difficult

An example: The Travelling Salesperson Problem

consider a *map* with N *cities* at given (irregular) positions $\left\{\vec{r}_i = (x_i, y_i)^T\right\}_{i=1,\ldots,N}$
and distances $d(i,j) \,=\, |\vec{r}_i - \vec{r}_j|$

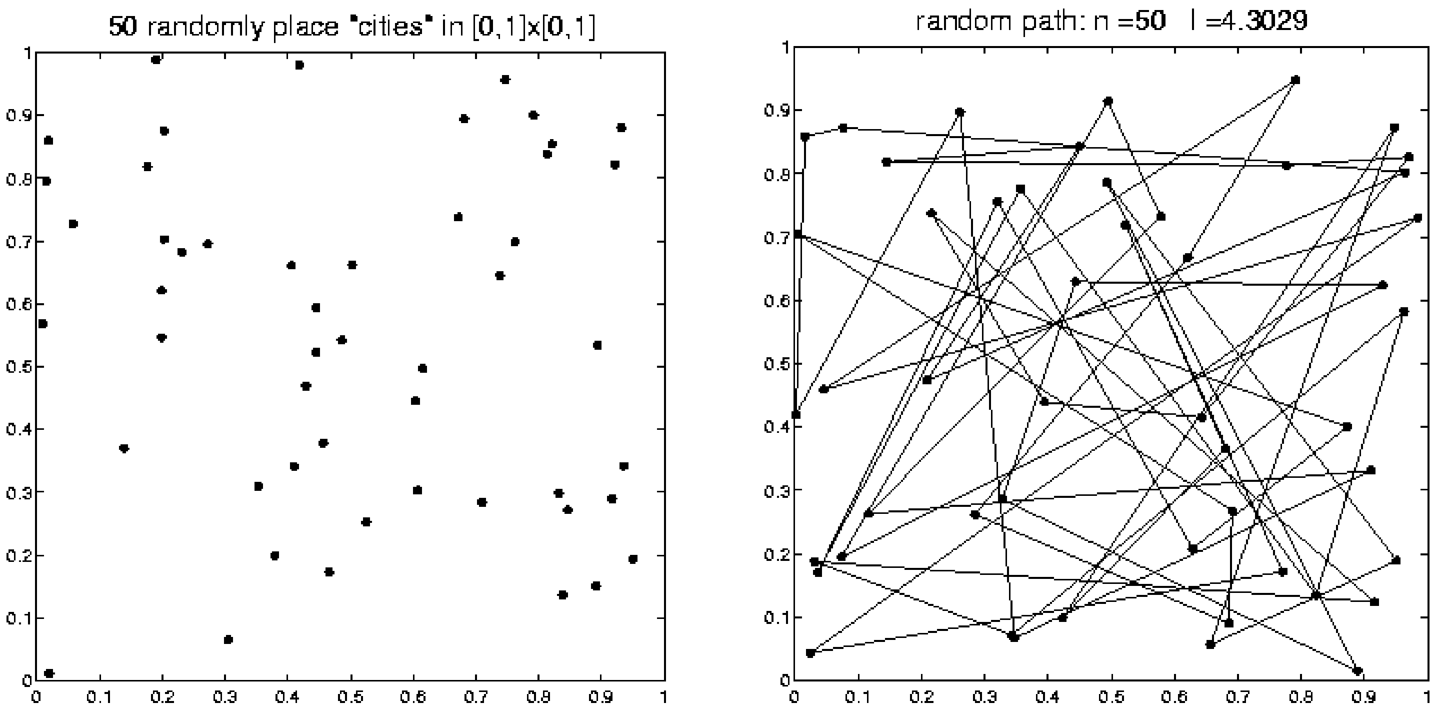


Fig. 1:
Left: *50 cities, at random positions in* $[0,1]\times[0,1]$
Right: *a path connecting all cities in randomized order*

Optimization Problem:
find the **shortest round trip** which visits every city exactly once

– the length ℓ of a trip is a function of the vector $\vec{o} = (o_1, o_2, \ldots, o_N)^T$
which contains each $i\,(1 \leq i \leq N)$ exactly once and defines the order
in which the cities are visited: $\ell(\vec{o}) \,=\, \sum_{k=1}^{N} d(o_k, o_{k+1}) + d(o_N, o_1)$

– for N cities, one expects ℓ to be on the order $\sqrt{N}$
$\rightarrow$ wherever numbers are given, they correspond to $\ell/\sqrt{N}$.

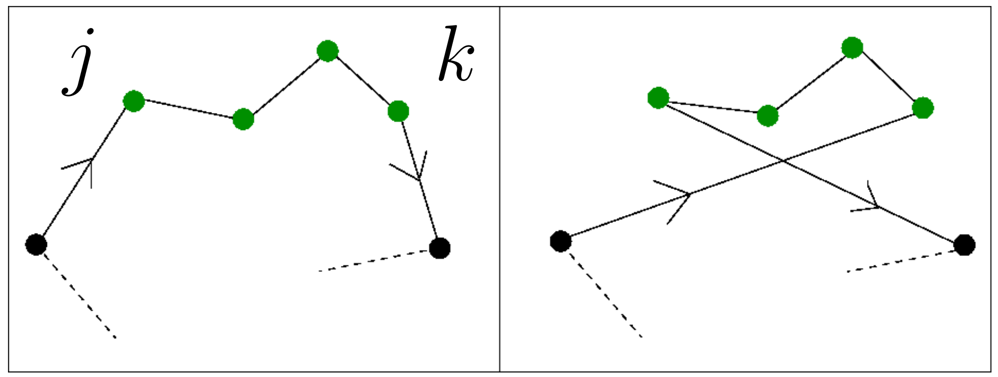
difficulties:

- the number of possible (different) paths is $(N-1)!/2$, grows like $\sim N^N$ for large N
  $\rightarrow$ exhaustive search of all paths is unrealistic for large N
- for given $\{\vec{r}_i\}$, the function $\ell(\vec{o})$ can have many local minima (see below)
  $\rightarrow$ *greedy* algorithms (steepest descent) will fail

Heuristic strategy: stochastic descent procedure

$0.$ start with a random path $\vec{o}(0)$ with length $\ell(0)$

$1.$ given $\vec{o}(t)$ with $\ell(t)$, suggest a *small* change, e.g.:
select two cities j and k randomly
cut the segment of the path between j and k
re-insert the segment in reverse order



here, ℓ increases from left to right

$2.$ calculate the potential new path length $\ell(t+1)$
if $\ell(t+1) < \ell(t)$ accept the move **(descent algorithm)**
else, let the path unchanged

iterate 1,2 until the path does not change anymore

Matlab code: stochastic optimization (TSP)

```
   function anneal = tsp(n,maxsteps,temp,met)
%  tsp(n,ms,temp,method) tries to find the shortest path
%  that connects n randomly placed cities
%  method=1 (2) corresponds to Metropolis (threshold) algorithm
%  ms*100 is the total number of performed steps
%  temp is the initial temperature, after each 100 steps it
%  is decreased by 1%.

   temps = temp;   % intial temperature
   lt = zeros(1,ceil(maxsteps));
   tt = 1:ceil(maxsteps);

   close all;
%  initialize random number generator and draw coordinates
   rand('state',0); cities = rand(n,2);
   order = [1:n];  op = path(order,cities);

   for jstep=1:ceil(maxsteps);
%  lower temperature by 0.1 percent
   temp = temp*0.999;
   for ins = 1:100
      j = ceil(rand*n); len = ceil(rand*(n/2));
      cand = reverse(order,j,len);
%  evaluate change of path length
      diff = delta(order,cities,j,j+len);
      np   = op + diff;
%  met=1: threshold, met=2: metropolis
      if ( (met==1 && (rand<exp(-diff/temp))||(diff<0)) || ...
           (met==2 && diff<temp))
         order = cand;
         op = np;
      end
   end

%  rescale length of path by sqrt(n) for output purposes
      lt(jstep) =  op/sqrt(n);
      curlen = path(order,cities)/sqrt(n);
%  plot map, cities and path
      figure(1); plotcities(order,cities);
      title(['n =',num2str(n,'%3.0f'),        ...
             '   t =',num2str(jstep*100,'%8.0f'),  ...
             '   l =',num2str(curlen,'%4.4f'),  ...
             '   T =',num2str(temp,'%6.6f')],   ...
             'fontsize',16);
     if (met==1)
        xlabel(['Metropolis algorithm, annealing'],'fontsize',16);
     else
        xlabel(['Threshold algorithm', ...
                '    T(0)=',num2str(temps,'%4.4f')], ...
                'fontsize',16);
     end
    end
%  plot evolution of length versus iteration step
      figure(2); plot(0,0); hold on;
      plot(tt,lt,'k.');
      title(['n =',num2str(n,'%3.0f'),        ...
             '   l =',num2str(curlen,'%4.4f'),  ...
             '   T =',num2str(temps,'%4.4f')],   ...
             'fontsize',16);
      if (met==1)
         xlabel(['Metropolis steps / 100'],'fontsize',16);
      else
         xlabel(['Threshold steps /100'],'fontsize',16);
      end
         ylabel(['l'],'fontsize',16);

---------------------------------------------------------------
```

```
  function t = path(order,cities)
% function path(...) returns the length
% of the trip when visiting the cities
% in the order specified by order

  trip = 0;
% cities 1 to n  and n to 1
  for i=1:length(order)-1
      piece = cities(order(i+1),:)-cities(order(i),:);
      trip  = trip +  sqrt(piece*piece');
  end
      piece = cities(order(1),:)-cities(order(length(order)),:)
;
      trip  = trip +  sqrt(piece*piece');

  t = trip;
```

```
  function diff = delta(order,cities,j,k)

% function delta(...) determines the change
% of the path length when the sub-path between
% j and k is reversed

   nn = length(order); jj = j; kk=k;
   jl = jl-1; kr = kk+1;

   if (kk>nn)
    kk = kk-nn;
   end
   if (jl ==0)
      jl=nn;
   end
   if (kr ==nn+1)
      kr=1;
   end
      piece = cities(order(jl),:)-cities(order(jj),:);
      p1 = sqrt(piece*piece');
      piece = cities(order(kr),:)-cities(order(kk),:);
      p2 = sqrt(piece*piece');
      piece = cities(order(jl),:)-cities(order(kk),:);
      p1new = sqrt(piece*piece');
      piece = cities(order(kr),:)-cities(order(jj),:);
      p2new = sqrt(piece*piece');

    diff = p2new- p2 + p1new - p1;
```

```
    function kt = plotcities(order,cities)
% function plotcities(order,cities) displays
% the cities and the current path in [0,1]x[0,1]

    ordcit = cities; nn = length(order);
    citx = zeros(nn+1,1); city = zeros(nn+1,1);

    hold off; plot(0,0); box on;
    axis square; hold on;

    for i=1:nn
     citx(i) = cities(order(i),1);
     city(i) = cities(order(i),2);
    end
     citx(nn+1) = cities(order(1),1);
     city(nn+1) = cities(order(1),2);

    plot(citx,city,'k.','MarkerSize',15);
    plot(citx,city);
    figure(1);
```

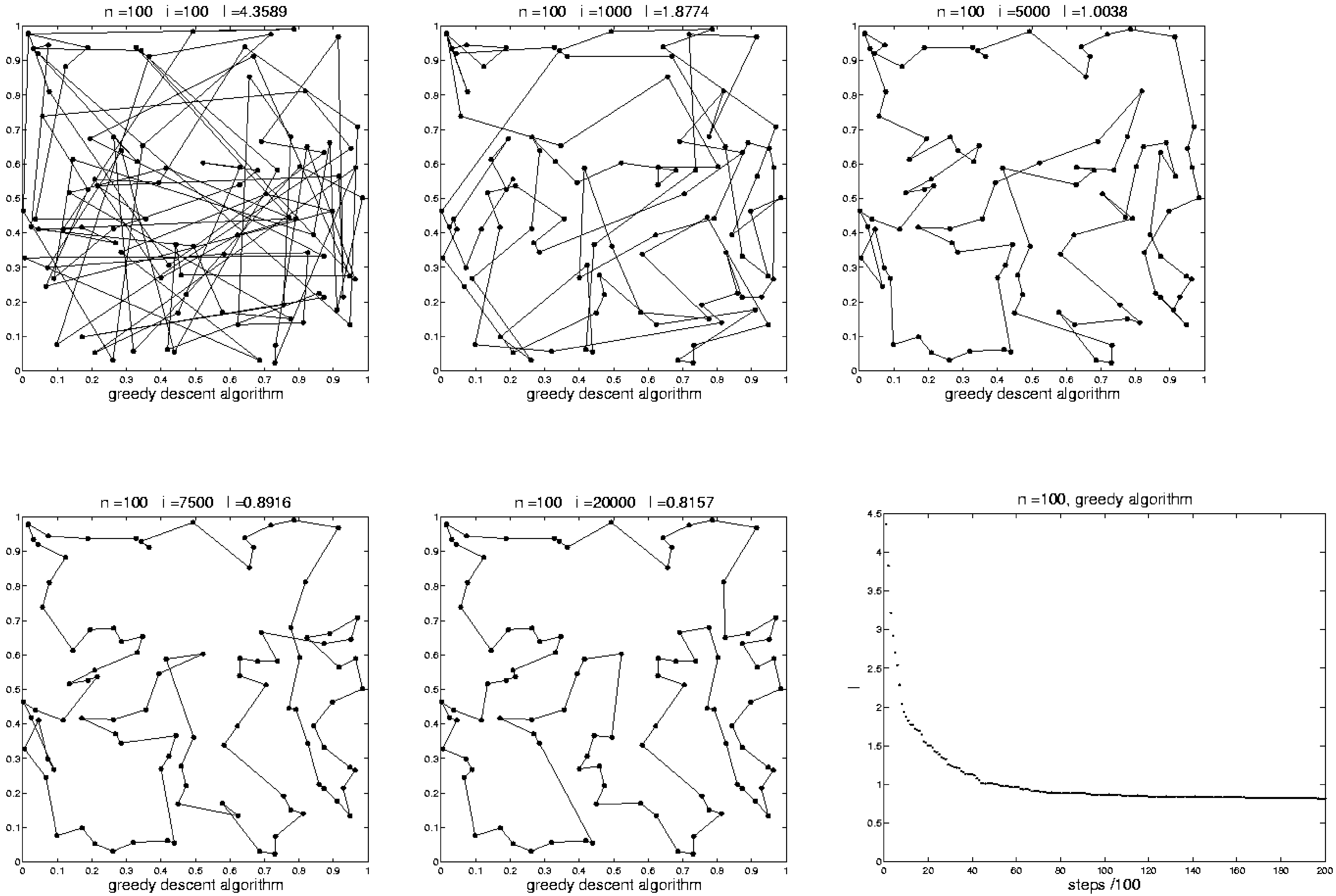


Fig. 4: *TSP with 100 random cities; evolution of the path when applying the descent algorithm. After about 20000 steps a stationary configuration is found. The lower right graph displays the monotonic decrease of* ℓ.

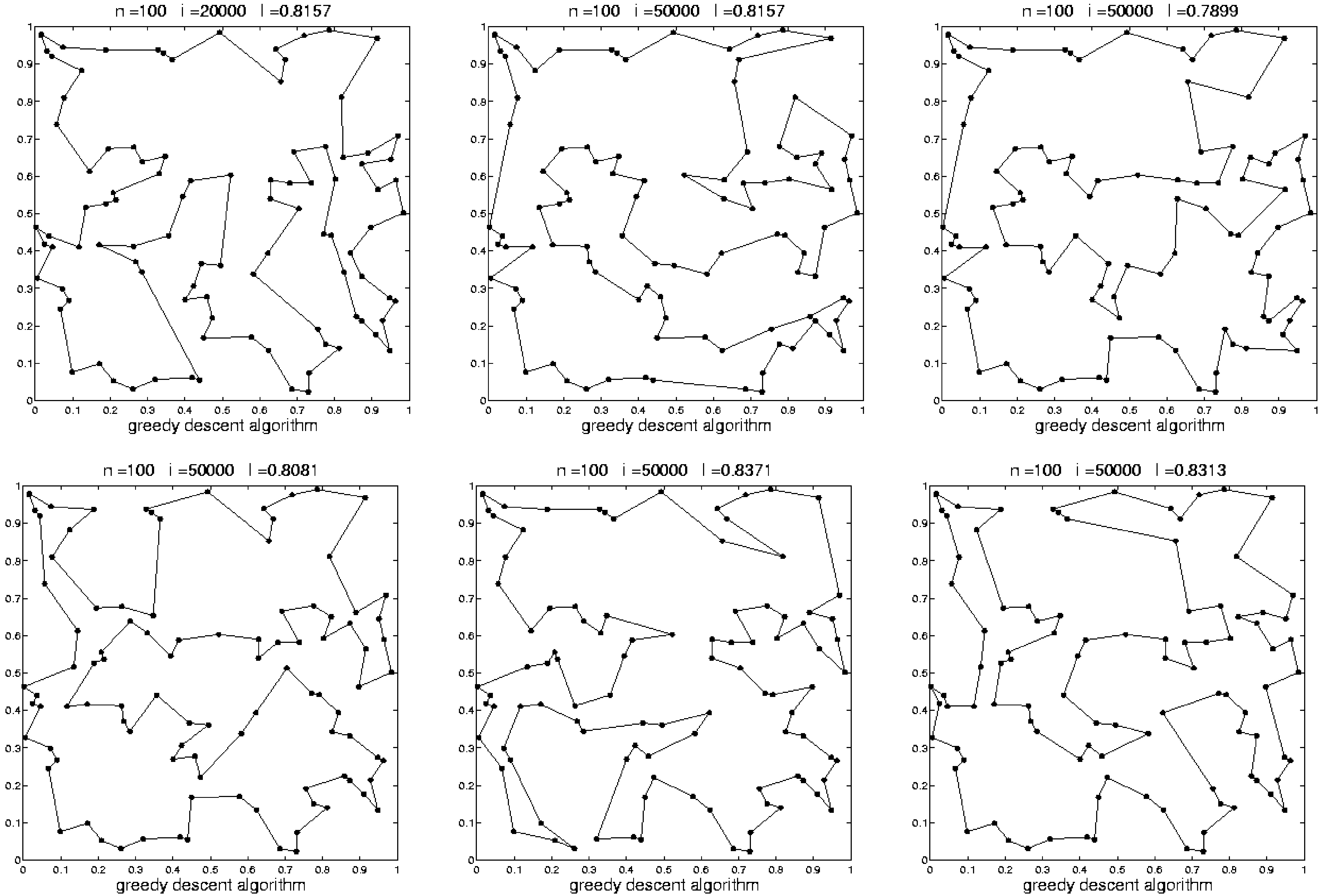


Fig. 6: *Stationary paths (not verified) of the descent alg., i.e. local minima, for the same set of cities, but with different initialization of the randomized search procedure. Although the values of ℓ are similar, paths can differ significantly.*

idea: allow for changes that (temporarily) increase $\ell(\vec{o})$
hope: algorithm can leave local minima and get closer to the global minimum

Metropolis algorithm : (probabilistic acceptance, $\leftarrow$ statistical physics)

$\ldots$
$2.$ calculate the potential new path length $\ell(t+1)$ and $\Delta\ell = \ell(t+1) - \ell(t)$
if $\Delta\ell < 0$, accept the move
else, accept it with a probability $P(\Delta\ell) = \exp(-\Delta\ell/T)$
$\ldots$

The **temperature parameter** $\boldsymbol{T}$ controlls the acceptance of $\Delta\ell > 0$:

$\boldsymbol{T = 0}$	only decreasing moves are accepted, descent version
$\boldsymbol{T \to \infty}$	all moves are accepted, no minimization at all

Note:
The Metropolis algorithm is frequently used to simulate physical systems at *real* temperature T, see next section.

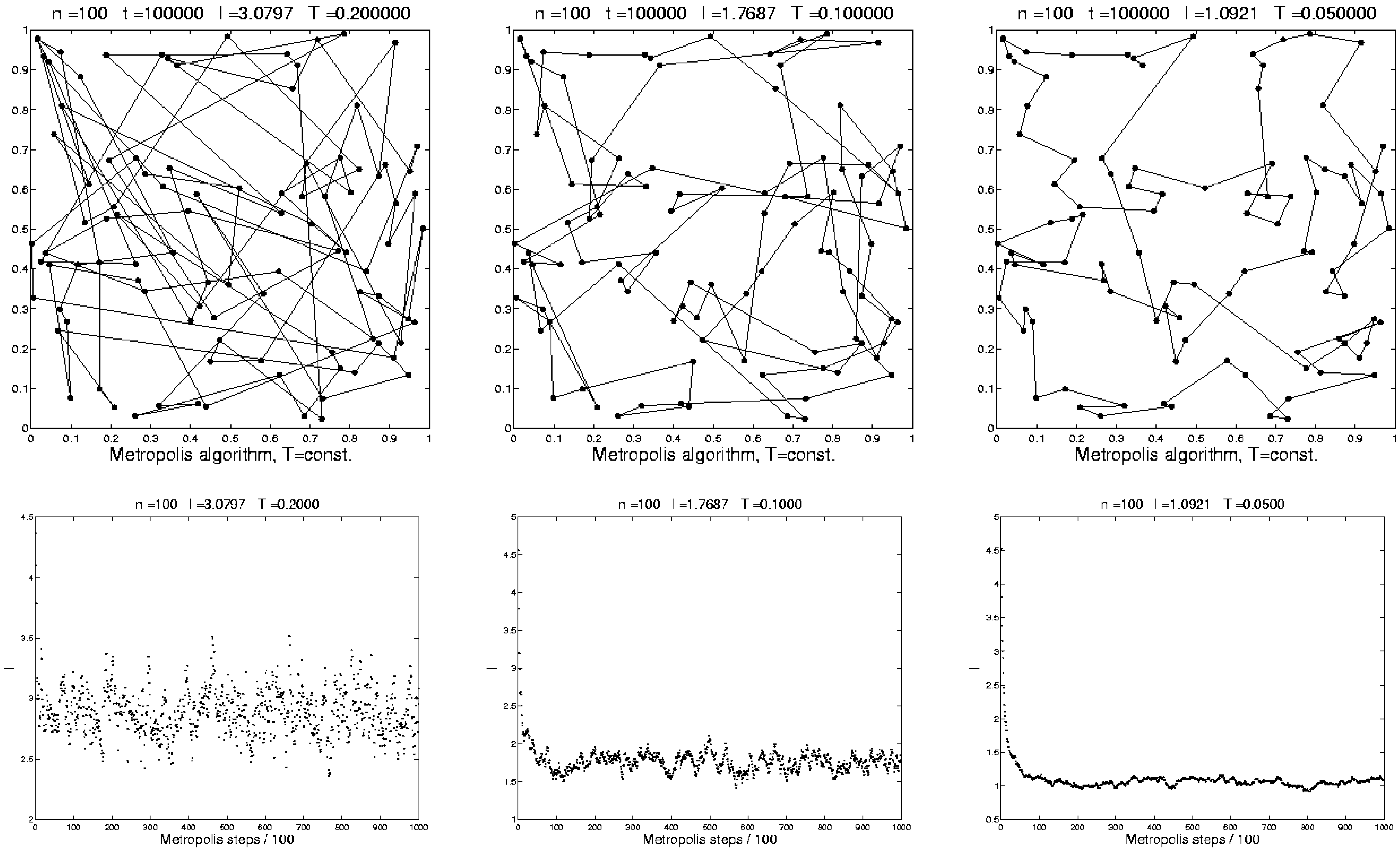


Fig7: Metropolis algorithm with $T = const$. Example paths after 100000 iteration steps (upper) and evolution of ℓ (lower). The temperature controlls the *equilibrium* value of ℓ, also fluctuations decrease with decreasing T. Note the different scalings of ℓ-axes in the plots.

Simulated Annealing

– start with relatively high temperature, large fluctuations

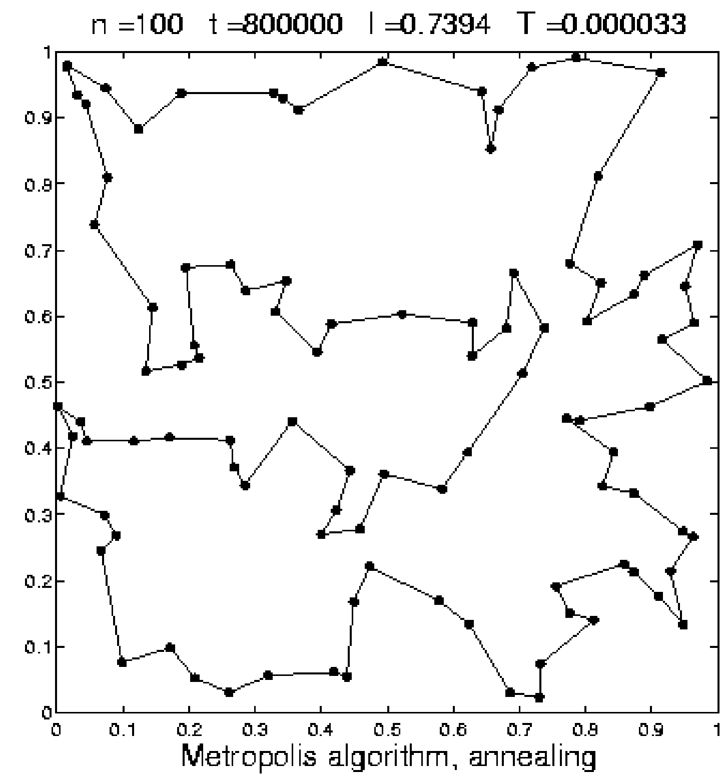
– decrease T during the iterations, i.e. *cool down* or *anneal* the system

problem: the decrease of T must be

– slow enough, so that local minima can be left

– fast enough to achieve solutions in realisting computing time

attempt: start with, say, $T = 0.1$,   after each 100 steps, reduce T by 0.1%



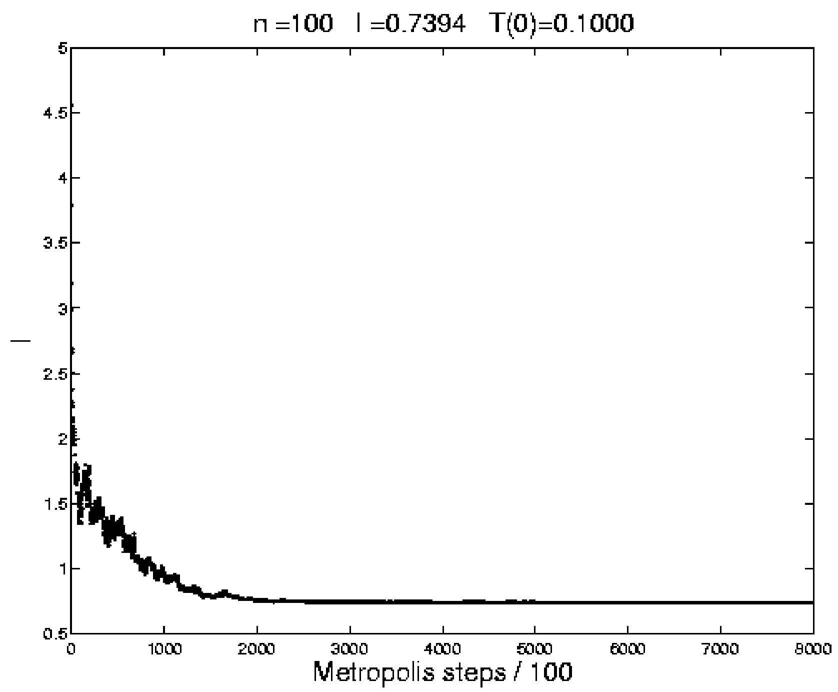


Fig. 8: *Annealing procedure for $N = 100$ cities, starting temperature $T = 0.1$. Left: the path after 800000 steps. Right: the evolution of ℓ in the iteration. Note that the achieved value is significantly lower than for the descent algorithm ($T = 0$).*

Deterministic acceptance:
simpler (faster) rule for acceptance of moves:

$\ldots$

$2.$ calculate the potential new path length $\ell(t+1)$ and $\Delta\ell = \ell(t+1) - \ell(t)$

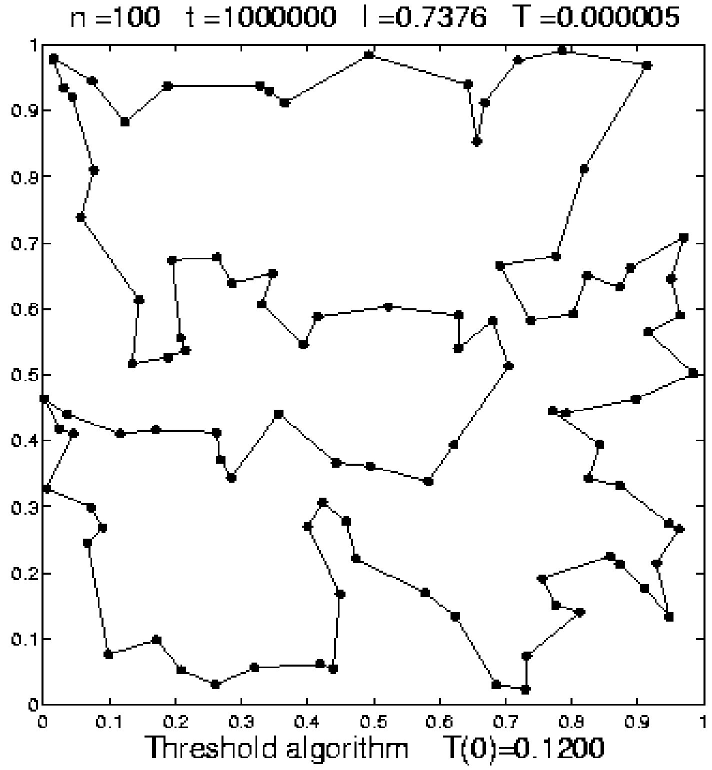
if $\Delta\ell < T$, accept the move

$\ldots$

– requires less computational effort (no random number, no `if(...)`)

– yields (here) comparable or better results, if T is reduced as in simulated annealing

– is not as well-founded theoretically; for instance, T is not a *temperature*



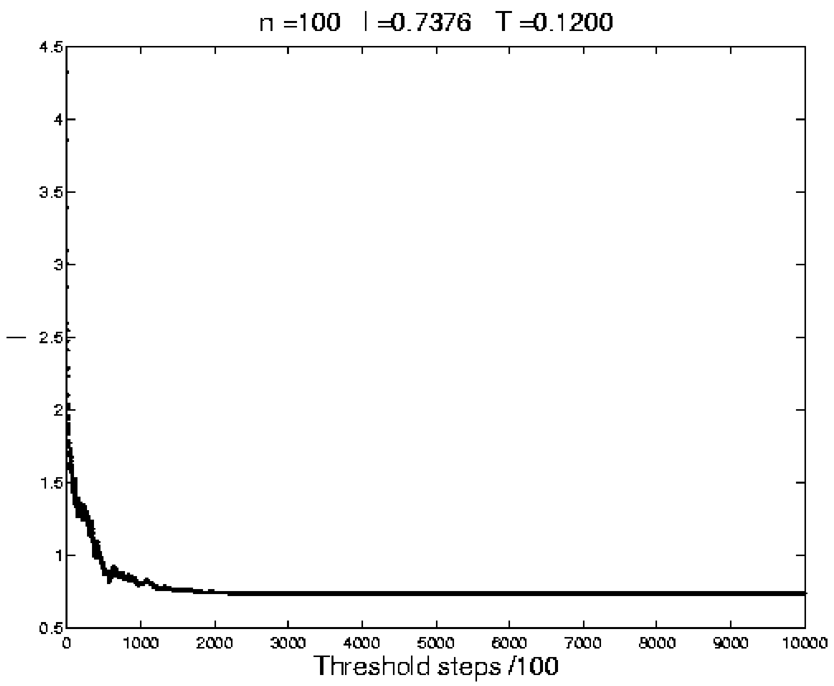


Fig. 9: *Threshold algorithm for* $N = 100$ *cities, starting threshold was* $T = 0.12$*. Left: the path after 100000 steps. Right: the evolution of* ℓ*.*