北京大學

数据结构与算法论文

模拟退火算法

课程名称：《 数据结构与算法 》

开课学期：2016—2017 学年度第一学期

报告作者：王旭斌

所在班级：地球与空间科学学院 2016 级

上交日期：2017 年3月

目录

模拟退火算法

王旭斌

（地球与空间科学学院 1600012630）

摘要：在处理一些 NP 问题时，随机算法的效率要优于枚举或启发式搜索等算法。模拟退火算法，作为一种来源于固体退火原理的算法，是常见的随机算法。本文从模拟退火算法的原理出发，分析了模拟退火算法的优点等，并通过实例说明其高效性。

关键词: NP；随机算法；模拟退火；最优解;；稳定性。

一、模拟退火算法的引入

对于生活中常见的最优决策类问题，最自然的想法就是枚举法。但对于一些数据规模较大的情况，受时间复杂度的限制，枚举法并不能在有效时间给出有效解[1]。于是有了动态规划（Dynamic-Programming），深度优先搜索（Deep-First-Search）的剪枝优化等。然而对于一些 NP 问题，例如 TSP 问题，常规方法的表现不甚优秀，甚至有时效率及其低下。随着需求的增加，随机算法应运而生，模拟退火算法便是其中比较优秀的一种。

二、模拟退火算法的介绍

1. 基本概念

模拟退火算法（Simulate-Anneal，简称 SA 算法）来源于固体退火原理：将固体加温至充分高，再让其徐徐冷却，加温时，固体内部粒子随温升变为无序状，内能增大，而徐徐冷却时粒子渐趋有序，在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。

根据 Metropolis 准则，粒子在温度 T 时趋于平衡的概率为 $e^{-\Delta E/kT}$，其中 e 为温度 T 时的内能，ΔE 为其改变量，k 为 Boltzmann 常数[2]。用固体退火模拟组合优化问题，将内能 E 模拟为目标函数值 f,温度 T 演化成控制参数 t，即得到解组合优化问题的模拟退火算法。

2. SA 算法的模型

第一步：获得一个初始解，并确定一种变换方式以衍生新解，变换方式需具有较强随机性，并尽可能覆盖多的初始解的邻域结构;

第二步：获得新解，并计算估价函数差值;

第三步：判断是否接受新解，新解更优则接受新解，否则以 Metropolis 准则接受新解;

第四步：更新当前解，并迭代。

流程图见附录一

三、算法分析

对于寻求一个问题的最优解或有效解，最直观的方法就是枚举法。枚举法虽然一定能获得最优解，但对于数据规模较大的问题往往不能在有效时间内获得。

一种改进方法是完全随机化搜索

[1] 有效时间：指人们为了获得该决策而乐意付出的时间；有效解：最优或比较接近最优解的一个方案。

[2] Boltzmann 常数：物理常数，在 SA 算法中可为自行规定的某常数值。

所有可能解，通过比较已被随机到的解而获得它们中的最优解。这种方法虽然省时，但它获得的解往往令人无法接受，即在有效时间内无法获得有效解。

于是爬山算法（Hill-Climbing）应运而生。爬山算法的原理是随机获得一个初始解，通过一定的变换规则[3]不停对当前解进行迭代，每次迭代更新当前解使其价值最高。爬山算法原理参见图 1。

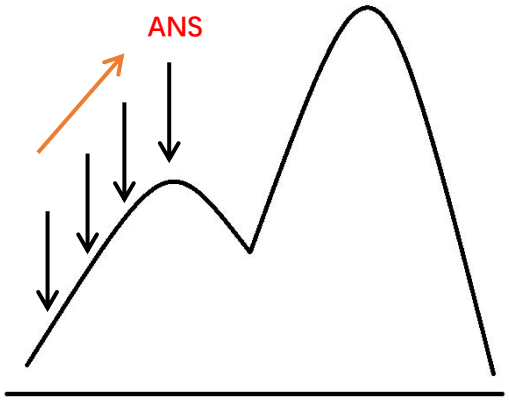


图 1 爬山算法的原理

从图中我们可以看出，爬山算法的效率很高，能在有效时间内获得有效解，然而这一算法的缺点是容易陷入局部最优解而忽略全局最优解，从而导致决策的失误。

这时，一种很自然的想法就是有一定概率接受更差的解作为当前解，从而跳出局部最优解。于是，人们选择 Metropolis 准则作为接受局部更差解的概率，产生了模拟退火算法。模拟退火算法原理参见图 2。

同样从图中我们可以看到，模拟退火算法有很大概率跳出局部最优解而向全局最优解靠拢，从而获得收益更高的解。

[3] 往往是基于随机化的变换规则

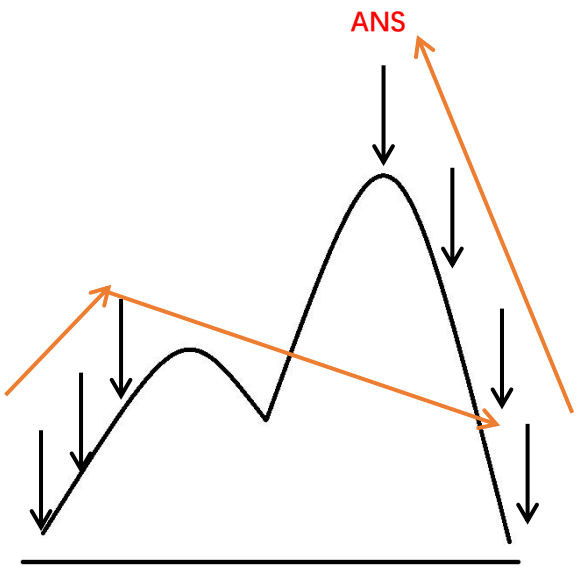


图 2 模拟退火算法的原理

四、应用实例（TSP 问题）

先介绍一下 TSP 问题(翻译作货郎担或旅行商问题）：从起点出发，对每个城市拜访且仅拜访一次，最后回到出发城市。

TSP 问题被认为是 NP 问题[4]，即不能在多项式时间内进行求解。借助于 SA 算法，我们可以在有效时间内获得有效解。

这里笔者借助 Python 分别使用枚举、随机、爬山、模拟退火四种方法求解了 TSP 问题（代码见附录 2），并附上测试结果表格（见附录 3）。

从测试结果中我们可以看出，综合考虑时间与获得解的能力，模拟退火算法要远远优秀于其他几种算法。

五、算法的优化

在不断的打代码、修改的过程中，笔者发现影响模拟退火算法的主要因素包括：初始与结束温度的选取，温度降低率，公式中 Boltzmann 常数的选取，变换规则（对第三组数据进行测试，测试结果见附录四）。

[4] 至少大多数人这样认为

1. 两个温度的选取

首先两个温度与温度降低率共同决定了循环执行的次数，适当拉大温度差有助于执行更多次循环而获得最优解。

另一方面，T 参与 Metropolis 准则中概率的计算。始终 T 所在的区间会间接影响到概率的变化，但考虑到我们对 T 的需要往往是体现温度的变化而影响概率的走势，并且概率的主要影响因素是公式中 Boltzmann 常数选取，所以这一方面的影响属于次要影响。

2. 温度降低率

温度降低率只从循环次数以及概率下降的速度上影响结果，影响的只是循环执行的次数。

3. Boltzmann 常数(k)的选取

该常数从概率大小上影响结果，我们认为在不同问题中放弃当前解而接受较差解的情况有所差异[5]，所以该常数的选取应根据实际情况决定，往往对于多峰问题[6]应使该常数大些，同时也应保持与初始温度在数量级上的一定关系[7]。并且 k 越小，结果越容易陷入局部最优解。

4. 变换规则

选取一个明智的变换规则有助于获得最优解。对于变换规则我们的考虑是：尽可能快覆盖到所有的解，尽可能向更优的方向变换[8]。

在所列举这些影响因素中，对于前面举例的 TSP 问题，我们增大两个温度的温差，增大温度减低率，增大 Boltzmann 常数会使模拟退火算法的稳定性更高[9]。

六、结论

模拟退火算法作为一种随机算法在处理大规模数据问题，尤其是一些 NP 问题时具有极强的优越性。在实际使用中应当按照实际情况设置算法中的参数以获得最大收益。

七、参考文献

[1] 魏延,谢开贵.模拟退火算法[J].蒙自师范高等专科学校学报,1999,1(4):7-11.

[2] 张建航,李国.模拟退火算法及其在求解 TSP 中的作用[J].现代电子技术,2006,22:157-158

[3] 范晔，周泓.作业排序模拟退火算法影响因素分析和一种多次淬火模拟退火法[J].系统工程理论方法应用,2003,11(1):72-76

[5] 比如一个问题在一个连续区间上是单调的并且不同解之间都可以通过互相衍生得到，那我们没有必要接受衍生出的更差的解。

[6] 指局部最优解不止一个的问题。

[7] 这时我们更关注的往往是最开始迭代接受较差解的概率。

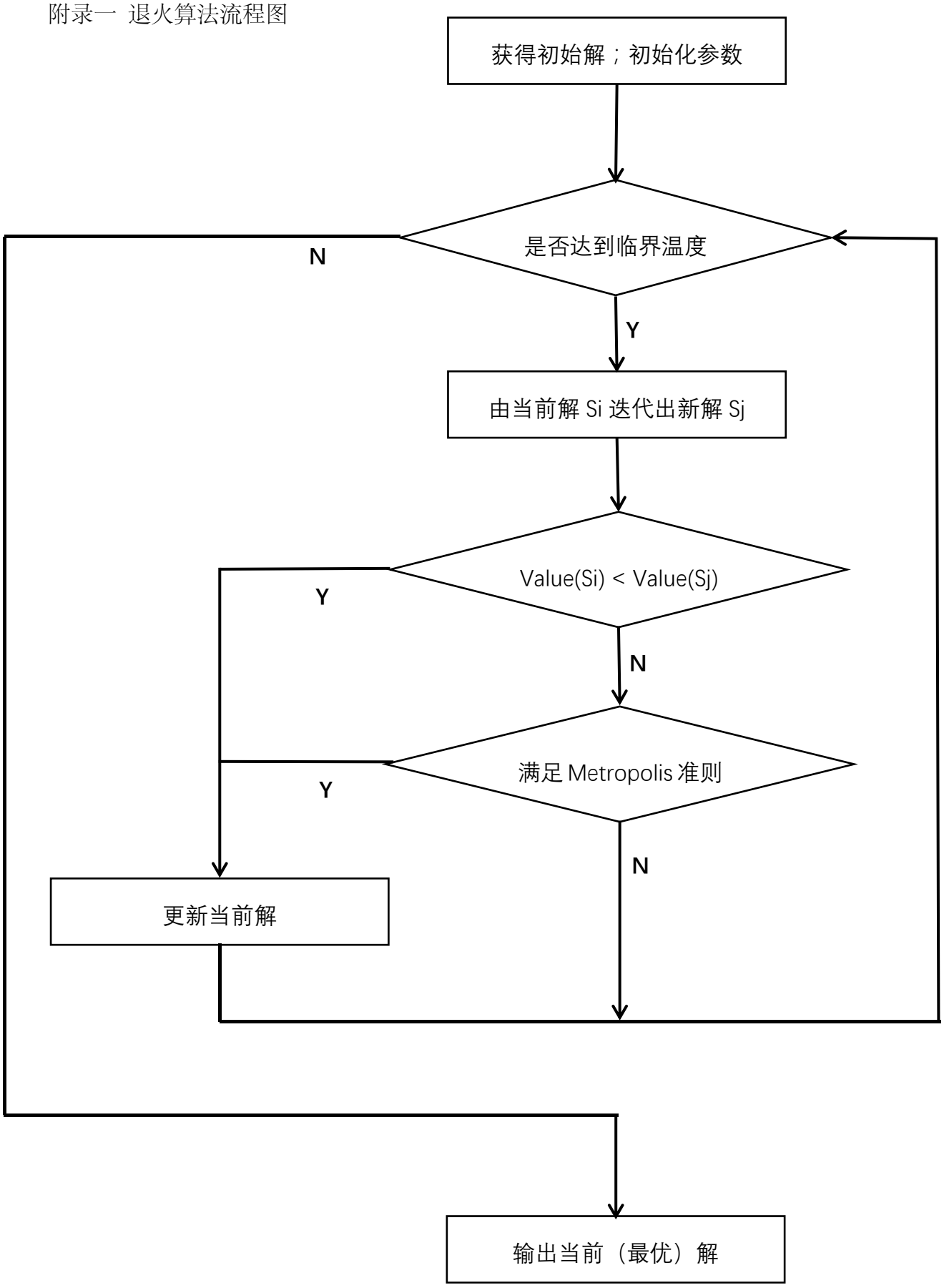
[8] 实际中往往无法判断更优的方向，所以一般希望获得一种或多种基于随机方法的变换规则。

[9] 多次运行的结果与最优解相差越小稳定性越高。


A
B
C
D


铁

附录一 退火算法流程图


获得初始解；初始化参数
是否达到临界温度
N
Y
由当前解 Si 迭代出新解 Sj
Value(Si) < Value(Sj)
Y
N
满足 Metropolis 准则
Y
N
更新当前解
输出当前（最优）解

附录二 涉及代码

Code No.1

```
# File Name: tsp-enum.py
import time

# Input
num = int(input())
file = open('data/input' + str(num) + '.txt', 'r')
n = int(file.readline())
dist = [[int(x) for x in file.readline().split()] for i in range(n)]
file.close()

# Time Record
StartTime = time.time()

# Initialization
global ans
vis = [False for i in range(n)]
ans = -1

#Dfs
def dfs(x, cnt, tot):
    vis[x] = True
    if cnt == n:
        global ans
        if tot + dist[x][0] < ans or ans == -1:
            ans = tot + dist[x][0]
    else:
        for i in range(n):
            if not vis[i]: dfs(i, cnt + 1, tot + dist[x][i])
    vis[x] = False

dfs(0, 1, 0)
print(ans)

# Time Record
EndTime = time.time()
print(EndTime - StartTime)
```

Code No.2

```python
# File Name: tsp-rand.py
import random
import time

# Input
num = int(input())
file = open('data/input' + str(num) + '.txt', 'r')
n = int(file.readline())
dist = [[int(x) for x in file.readline().split()] for i in range(n)]
file.close()

# Time Record
StartTime = time.time()

# find a path randomly and find 10^5 times
# Good Luck!
ans = -1
for j in range(100000):
    l = [i for i in range(1, n)]
    pre = tot = 0
    for i in range(n - 2, -1, -1):
        tmp = l[random.randint(0, i)]
        tot += dist[pre][tmp]
        pre = tmp
        l.remove(tmp)
    tot += dist[pre][0]
    if tot < ans or ans == -1: ans = tot
print(ans)

# Time Record
EndTime = time.time()
print(EndTime - StartTime)

```

Code No.3

```python
# File Name: tsp-hc.py
import random
import time

```

```
# Input
num = int(input())
file = open('data/input' + str(num) + '.txt', 'r')
n = int(file.readline())
dist = [[int(x) for x in file.readline().split()] for i in range(n)]
file.close()

# Time Record
StartTime = time.time()

# Initial Path
t = [i for i in range(1, n)]
d = [0]
for i in range(n - 2, -1, -1):
    tmp = t[random.randint(0, i)]
    d.append(tmp)
    t.remove(tmp)
ans = dist[d[-1]][d[0]]
for i in range(1, n): ans += dist[d[i - 1]][d[i]]

# Hill-Climbing
# How to change the available answer
# find Left_bound and Right_bound
# reverse the nodes between Left_bound and Right_bound
cnt = 0
while True:
    l = random.randint(1, n - 1)
    r = random.randint(1, n - 1)
    if l == r: continue
    elif l > r: l, r = r, l
    delta = dist[d[l - 1]][d[l]] + dist[d[r]][d[(r + 1) % n]] -\
            (dist[d[l - 1]][d[r]] + dist[d[l]][d[(r + 1) % n]])
    if delta > 0:
        ans -= delta
        tmp = d[l: r + 1]
        tmp.reverse()
        d = d[0: l] + tmp + d[r + 1: n]
        cnt = 0
    else:
        cnt += 1
```

```
45          if cnt > 10000: break
46  print(ans)
47
48  # Time Record
49  EndTime = time.time()
50  print(EndTime - StartTime)
51
```

Code No.4

```
1   # File Name: tsp-sa.py
2   import random
3   import math
4   import time
5
6   # Input
7   num = int(input())
8   file = open('data/input' + str(num) + '.txt', 'r')
9   n = int(file.readline())
10  dist = [[int(x) for x in file.readline().split()] for i in range(n)]
11  file.close()
12
13  # Time Record
14  StartTime = time.time()
15
16  # Initial Path
17  t = [i for i in range(1, n)]
18  d = [0]
19  for i in range(n - 2, -1, -1):
20      tmp = t[random.randint(0, i)]
21      d.append(tmp)
22      t.remove(tmp)
23  ans = dist[d[-1]][d[0]]
24  for i in range(1, n): ans += dist[d[i - 1]][d[i]]
25
26  # Simulated-Annealing
27  # T -> Initial Temperature
28  # MinT -> Terminal Temperature
29  # delta -> Reduce T
30  # k -> constant
31  # Probability = exp(dE / (k * T))
```

```
# Changing Method
# reverse the nodes between Left_bound and Right_ bound
T = 10.0 ** 11
MinT = 1.0 ** -6
delta = 0.9999
k = 10.0 ** 10
while T > MinT:
    l = random.randint(1, n - 1)
    r = random.randint(1, n - 1)
    while l == r:
        l = random.randint(1, n - 1)
        r = random.randint(1, n - 1)
    if l > r: l, r = r, l
    dE = dist[d[l - 1]][d[l]] + dist[d[r]][d[(r + 1) % n]] -\
          (dist[d[l - 1]][d[r]] + dist[d[l]][d[(r + 1) % n]])
    if dE > 0 or (dE <= 0 and random.random() <= math.exp(dE / T)):
        ans -= dE
        tmp = d[l: r + 1]
        tmp.reverse()
        d = d[0: l] + tmp + d[r + 1: n]
    T *= delta
print(ans)

# Time Record
EndTime = time.time()
print(EndTime - StartTime)

```

Code No.5

```
# File Name: Data_Random.py
import random
num = int(input('Number of File:'))
file = open('data/input' + str(num) + '.txt', 'w')
n = int(input('Input N:'))
dist = [[random.randint(1, 10000) for i in range(n)] for i in range(n)]
file.writelines(str(n) + '\n')
for i in range(n):
    for j in range(n):
        if i == j: dist[i][j] = 0
        elif i < j: dist[i][j] = dist[j][i]
```

```
12            file.write(str(dist[i][j]) + ' ')
13        file.write('\n')
14    file.close()
15
```

说明:

Code No.1 tsp_enum.py 枚举算法

Code No.2 tsp_rand.py 随机算法

Code No.3 tsp_hc.py 爬山算法

Code No.4 tsp_sa.py 模拟退火算法

Code No.5 Data_Random.py 随机生成数据

附录三 测试结果

Data No.	1		2		3	
N=	10		20		30	
	输出结果	运行时间	输出结果	运行时间	输出结果	运行时间
tsp_enum.py	20540	0	None	＞1h	None	＞1h
tsp_rand.py	20540	2.215	47810	4.374	85357	6.724
tsp_hc.py	20540	0.047	25788	0.05	27231	0.053
tsp_sa.py	20540	1.574	19506	1.511	22428	1.458

备注：由于 Python 语言的特性以及枚举法自身的限制，笔者在电脑上没有获得枚举法在 2、3 组的解。

附录四、不同影响因素产生的结果

Test Case:	1		2		3	
N=	30		30		30	
	输出结果	运行时间	输出结果	运行时间	输出结果	运行时间
Initial	22428	1. 458	23166	1.478	23792	1.440
dT = 10^{11}	25107	1.448	23888	1.486	24640	1.395
delta = 0.99	33921	0.017	33152	0.015	32565	0.015
k = 10^{5}	22735	1.357	22471	1.367	24371	1.479
trans=change	24271	1. 215	24065	1. 107	23333	1. 195

备注：
后续几组测试数据对应程序中使用的参数均是根据第一组测试数据对应程序中的参数修改得到的。
Initial: dT = 10^{17}, delta = 0.9999, k = 10^{10}, trans = reverse

变量（变换规则）说明：
dT: 初始与结束温度的比值
delta: 温度降低率
k: Boltzmann 常数
trans: 变换规则
reverse: 选取一段并反转; change: 选取两个点并交换
保证比较当前解与当前衍生解的时间复杂度为 O(1)