

人工智能实验报告

遗传算法

小组 lgd-cs

赖启东* 14347058

黎丁嘉 15336077

李振宇 15336092

刘键涵 15336110

洪培街 15336059

2015级计算机科学与技术

School of Data and Computer Science, Sun Yat-Sen University, China

laiqd@mail2.sysu.edu.cn

December 26, 2017

Abstract

本次实验我们组采用遗传算法(Genetic Algorithm, GA)解决旅行商问题(Travel Saleman Problem, TSP)。我们在最朴素的遗传算法基础上加上少许改进, 拿数据ch130.tsp进行测试, 得出结果与最优解差距约为3.7%, 并与之前模拟退火算法对比。我们发现在这个数据上遗传算法表现出收敛快, 结果好的特点。

1 Introduction

1.1 Problem Description

给定一个无向图 $G = (N, E)$, 其中 $N = \{1, 2, \dots, n\}$ 是顶点集 $E = \{(i, j) : i, j \in N, i \neq j\}$ 是边集, 每条边 (i, j) 有一个正数表示旅行花费 $c[i, j]$ 。路径 $r = (v_1, v_2, \dots, v_n)$ 是一个 N 中每个点各出现一次的序列, 其中 $v_i (i \in [1, n])$ 表示序列中第 i 项。序列中第一项 v_1 是起点, 旅行商按照路径顺序访问每个点后回到起点, (最后从 v_n 回到 v_1)。路径 r 的花费 $cost_r = c[v_n, v_1] + \sum_{i=1}^{n-1} c[v_i, v_{i+1}]$ 。TSP就是给定这样一个无向图, 求一条花费最小的路径。

1.2 Algorithm Review

遗传算法是Holland在20世纪60年代末提出来的, 受遗传学中自然选择和遗传机制启发发展起来的一种搜索算法。

遗传算法将择优与随机信息结合起来, 是一个迭代过程。每次迭代中保留一组候选解, 并按照某种有些指标进行排序, 然后按照某种指标中选出一些解, 利用遗传算子, 进行运算产生新一代的一组解。新一代的解可能发生变异。重复迭代过程, 直到满足指定的收敛要求为止[1]。

2 Algorithm Design

2.1 Primal Algorithm

最经典最朴素的遗传算法流程图如图2.1。

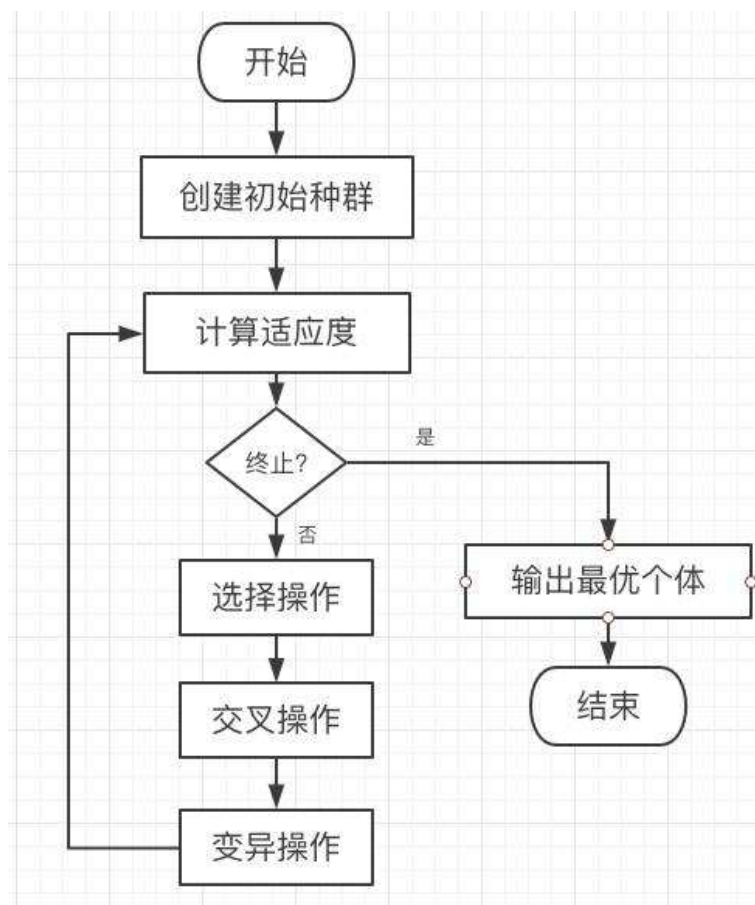


Figure 1: 图2.1

2.2 Our Approach

我们的算法基本框架与传统算法差不多，不同的是**增加了父代的变异操作**。因为通过实验发现，种群里的路径有可能是某个局部最优的解所产生的不同表示的序列。也就是从实验表现来看，通过交叉操作后再变异难以产生新的好解。局部最优一直保存在种群中，新生成的一代无法突破父辈，一直陷入局部最优。然而，加入父代的变异操作后，相当于对父代进行一次局部搜索，能够更好扩展出去得到优解。现实生活中也会有个体进行基因突变，产生好的基因，然后再遗传给下一代，这样整个种群在进化。所以这个父代变异操作也是合理的。

2.3 Configure

编码 一个个体的编码就是解本身，也就是一条路径。

适应值函数 记最优解的路径花费为 ans ,那么路径 r 的适应值为

$$f_r = \frac{1}{\frac{cost_r - ans}{ans} + 0.01} \quad (1)$$

可以理解为与最优解差的百分比的倒数。个体越好，其适应值函数值越大。

初始种群 使用`random_shuffle()`函数生成 $1 \sim n$ 的全排列作为一个初始个体，生成多个个体组成初始种群。

终止条件 设参数 $baditeration$,如果算法迭代超过 $baditeration$ 次没有更新最优解，那么姑且认为已经收敛，这时候结束程序。

选择 采用轮盘赌。令种群内个体数为 $pocnt$,第 i 个解(路径)的**相对适应值**

$$rf_i = \frac{f_{r_i}}{\sum_{j=1}^{pocnt} f_{r_j}} \quad (2)$$

生成一个 $[0,1]$ 的实数 x ,若

$$rf_1 + rf_2 + \dots + rf_{k-1} < x \leq rf_1 + rf_2 + \dots + rf_k \quad (3)$$

则选择个体 k 。

交叉算子 根据论文[2]启发所写(不完全一样)。交叉概率为 p_1 ，若不发生交叉则父代与子代一样，否则执行交叉算法。交叉算子时间复杂度为 $O(n)$ 。通过轮盘赌选择两个父代 r_i 和 r_j ，先随机产生一个数 $y \in [1, n]$,找 y 在 r_i 中的位置为 w_1 ,在 r_j 中的位置为 w_2 。

子代1 设置两个指针 $t_1 = w_1, t_2 = w_2$,一开始序列 $v = (y)$,然后进行 $n - 1$ 次选择。第 i (i 为奇数)次选择时不停左移指针 t_1 ，即 $t_1 = (t_1 - 1 + n) \% n + 1$ ，直到 r_i 的第 t_1 项不在 v 中(已经出现的话继续移动指针)，然后将其加入 v 的最左端;第 i (i 为偶数)次选择时不停右移指针 t_2 ，即 $t_2 = (t_2 + 1) \% n + 1$ ，直到 r_j 的第 t_2 项不在 v 中，然后将其加入 v 的最右端。 $n - 1$ 次结束后 v 成为一个新的子代。可以简单理解为 r_i 某点的左段与 r_j 对应点的右段连在一起成为新的个体。

子代2 按照生成子代1同样的方法，不过是 t_1 不停右移， t_2 不停左移，也可以生成一个新的子代。可以简单理解为 r_i 某点的右段与 r_j 对应点的左段连在一起成为新的个体。

example $r_i = (1, 2, 3, 4, 5), r_j = (3, 1, 5, 2, 4)$ 。随机值 $y = 4$,则可得子代1为 $(2, 3, 4, 1, 5)$ ，子代2为 $(1, 5, 4, 2, 3)$ 。

变异算子 子代与父代变异均用此算子。即2-opt算子，选取两个随机位置 w_1 和 w_2 ，然后将这条路径这两个位置截取的连续子序列翻转。例如路径是 $r = (5, 4, 3, 2, 1)$ ，随机的位置 $w_1 = 1, w_2 = 3$,则将5,4,3这一段翻转后，新路径 $r = (3, 4, 5, 2, 1)$ 。邻域大小为 $O(n^2)$ ，翻转一次时间复杂度为 $O(n)$ 。个体变异概率为 p_2 ，若发生变异则随机 cnt 次，从这 cnt 中选一个**花费最小**的个体作为变异结果。

保留 算子所产生的子代个体数量若大于 $pocnt$ ，则按适应值从大到小排序后保留适应值大的子代个体。

2.4 Solution Framework

以下是加入父代变异后的GA算法框架。其中 $f1$ 代表父代种群， $f2$ 代表子代种群。4是记录上一次迭代最优解，7是把子代变异。10是更新换代，旧子代变成新的父代，给下一次迭代。12是判断是否收敛。

Algorithm 1 GA(改进版)

```

1:  $f1 \leftarrow$  初始种群(初始解集)
2:  $iteration \leftarrow 0$ 
3: repeat
4:    $lastans \leftarrow ans$ 
5:    $f2 \leftarrow$  空集
6:    $f2 \leftarrow f2 \cup f1$ 交叉产生的解
7:    $f2 \leftarrow f2 \cup f2$ 变异产生的解
8:    $f2 \leftarrow f2 \cup f1$ 变异产生的解
9:    $f2 \leftarrow f2$ 的适应值前 $pocnt$ 大的个体
10:  交换 $f1, f2$ 
11:  用 $f1$ 里最优个体去更新 $ans$ 
12:  if  $ans$ 比 $lastans$ 好 then
13:     $iteration \leftarrow 0$ 
14:  else
15:     $iteration \leftarrow iteration + 1$ 
16:  end if
17: until  $iteration > baditeration$ 
18: return 最优解 $ans$ 

```

3 Computational Experiments

3.1 Parameter Settings

种群大小 $pocnt = 60$,变异时候2-opt搜索次数 $cnt = n(n$ 为点数)。交叉概率 $p_1 = 0.7$,变异概率 $p_2 = 0.3$ (父代子代均是),最坏迭代次数 $baditeration = 700$ 。

3.2 Datasets

选取的数据为ch130.tsp。这个数据里有130个点，分布较分散。

3.3 Environment

我们的程序使用C++语言编写，实验运行在Intel Core i7-4510U 2.0GHz的CPU，8GB RAM，Ubuntu 14.04LTS操作系统的个人电脑上，使用了g++O4优化。

3.4 Results

如图3.4(1)为进行测试的效果图。表1为对ch130.tsp进行测试的结果。图3.4(1)是某一次测试得出的结果。最左边显示的是遗传算法的效果;中间的是模拟退火得出的解;最右边

为官方提供的最优解(ch130.opt.tour)。

我们这个程序是把两个算法合在一起运行，边运行边显示结果，可以较好对比。随着迭代的进行，两个算法得出结果可以地看出优劣，与最优解的差距。使用画图可以直观看出来路线是否有交叉，哪些地方走法不一样，列出误差百分比可以量化与最优解的差距。

同时UI中还加入几个重要参数的设置窗口以及不同输入数据的接口。这样方便测试者针对不同数据进行参数调试，适应多种多样数据。内部算法均使用C++内置容器类型，可以适用于不同规模的数据。这使得我们的程序有较强鲁棒性。

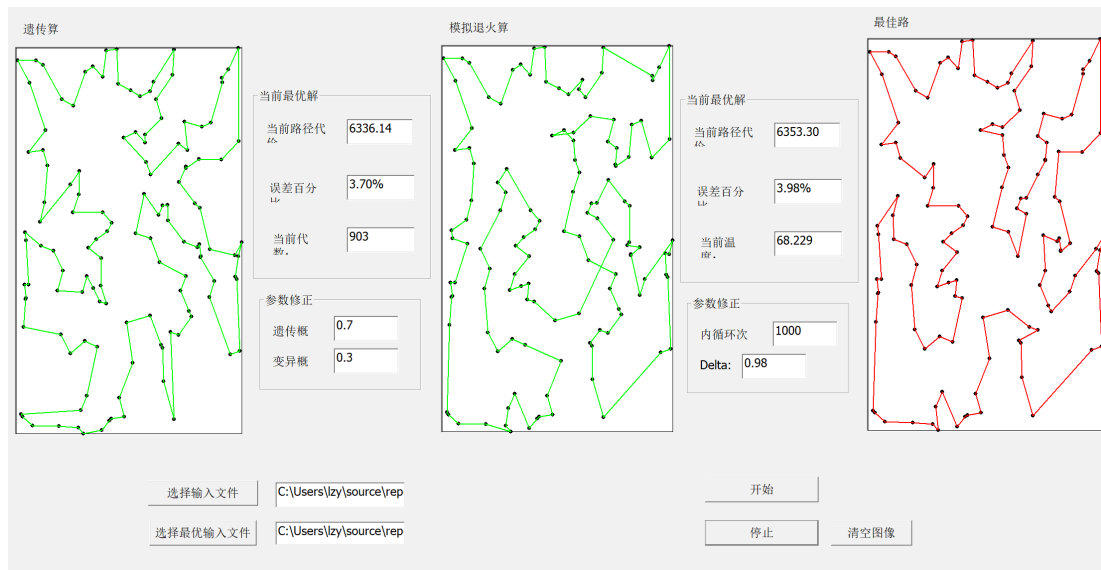


Figure 2: 图3.4(1)

如图3.4(2)为遗传算法随着迭代代数增加，每代最好解与官方最优解的误差百分比。大约到300代就收敛了。

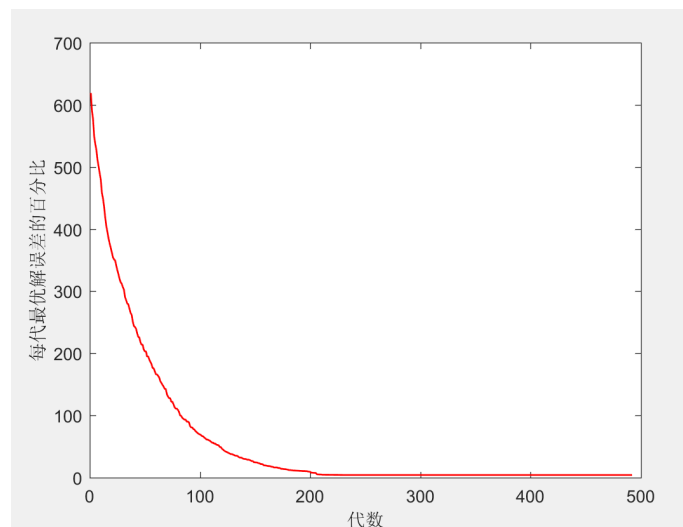


Figure 3: 图3.4(2)

Table 1: 测试结果				
测试次数	遗传算法		模拟退火	
	花费	时间	花费	时间
1	6336.14	1.86	6353.3	17.56
2	6609.63	1.84	6362.02	18.1
3	6361.8	1.88	6353.3	17.85
4	6388.2	1.88	6335.59	18.13
5	6453.39	1.85	6353.3	18.05
6	6674.9	1.83	6335.59	17.69
7	6429.12	1.87	6334.97	17.95
8	6580.85	1.85	6362.02	18.02
平均值	6479.254	1.8575	6348.761	17.91875
方差	13918.41	0.000294	118.819	0.036461
最优值	6336.14		6334.97	

表1中列出8次运行的结果，表中时间单位为秒。我们求出了均值与方差，并找出算法所得最优解。在表格中加粗的是该算法得出的最优解。

官方提供的最优解为**6110**。可以算出两种算法得出最好解与最优解的误差：

$$Error_{GA} = \frac{6336.14 - 6110}{6110} = 3.70\%$$

$$Error_{SA} = \frac{6334.97 - 6110}{6110} = 3.68\%$$

老师要求不超过最优解的10%，即 $6110 * (1 + 10\%) = \mathbf{6721}$ 。可以看出表中结果均小于这个值，说明算法稳定能满足老师的基本要求。

两种算法运行多次取最优都在4%内，说明两种算法都能得出较好的解，让人满意。

模拟退火 所得结果方差较小，比较稳定，但是用的时间比较长，搜索的解空间较大。因为初期接受劣解概率较大，所以保持了多样性，容易跳出局部最优解，代价是运行时间长。

遗传算法 所得结果方差较大，得出最优解结果挺好，用时少。因为我们实现的算法结构较为简单，没有很大力度保持物种多样性，所以收敛快，容易跳进局部最优。但由于算法本身思想优秀，所以还是可以得出较好的近似解。

4 Conclusion

通过这次实验，我们学到了遗传算法的基本框架，感觉遗传算法很奇妙。我们对其进行改进，并将运用到TSP问题上，发现遗传算法得出效果很好。通过与模拟退火算法对比发现它与模拟退火算法差不多，但觉得我们的遗传算法还有待改进。希望在接下来的学习实验之中有更多收获。

5 实验分工

赖启东完成GA算法核心部分。

李振宇将GA移植到MFC上并完成初版程序。

黎丁嘉优化界面，dubeg。

洪培銓优化算法，加快速度。

刘键涵调试程序，调整算法参数，提高最优解。

实验报告由赖启东撰写，平时实验过程记录报告由刘键涵完成。

References

- [1] 朱福喜, “人工智能基础教程(第二版),” 2011.
- [2] H. Sengoku and I. Yoshihara, “A fast tsp solver using ga on java,” 1998.