《高级运筹学实验报告》

遗传算法解决 TSP 问题

Ben

日期：2019.6.9

一、实验目的

改进和实现遗传算法，用以对 TSP 问题进行建模和近似求解，从而深入对启发式算法的理解。

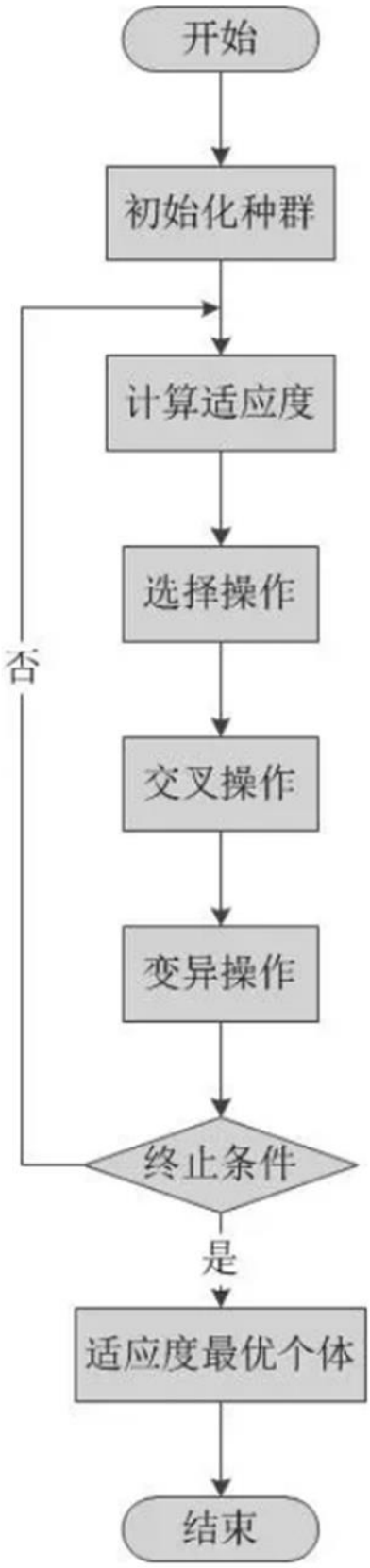
二、实验说明

本实验采用的编程语言为 python3，参与计算的 CPU 型号为 Core(TM) i7-4720HQ，实验数据来源为 TSPLIB，以及网站 http://www.math.uwaterloo.ca/tsp/index.html。

三、算法设计。

3.1 算法流程。

遗传算法解决 TSP 的流程是以下几部分：初始化种群、计算适应度函数、选择、交叉、变异然后不断重复直到找到理想的解。


开始
初始化种群
计算适应度
选择操作
交叉操作
变异操作
终止条件
否
是
适应度最优个体
结束

3.2 模型设定。

（1）种群初始化。

需要设定的参数是随机生成的初始解的数量，该数量过少会导致种群多样性不足，数量过多会降低算法的效率，我们设定种群规模（初始解数量为 150)。

（2）适应度函数。

根据数据集说明，其最优解采用的边权重类型为：EDGE_WEIGHT_TYPE : EUC_2D，即两城市之间的距离通过欧式距离计算。

$$distance_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

我们得到对路径的所有距离进行求和得到 distance，令 f=1/distance，即为适应度函数。

（3）选择

选择，即在上一代生存的个体中，通过优胜劣汰，使适应性更强的解得以保留。具体而言首先将上一代种群中适应性最强的 10%物种保留，然后通过轮盘转赌法，以选择概率为权重，挑出剩下的 90%物种。其中对于每个物种s_i）选择概率计算公式为：

$$p(s_i) = \frac{f(s_i)}{\sum_{i=1}^{n} f(s_i)}$$

采用上述设定的原因是尽量让适应度更强的物种活下来，同时防止适应性最强的物种因随机性而被轮盘转赌法淘汰。

（4）交叉

通过选择幸存下的物种进行交叉的概率为 70%，交叉的方式为单点交叉，即随机选取一个节点，将交叉双方该节点后的部分进行交换。在

交换后，单个物种可能会出现有重复城市的情况，因此我们进行了去重操作，即记录下重复的位置，使交叉双方重复的节点进行交换。

（5）变异

变异是遗传算法跳出局部最优解的重要操作。在 TSP 问题中，变异操作是随机选取物种的两个节点，将节点中的城市顺序颠倒。过往的研究表明，变异的概率大于 0.5 之后，遗传算法将退化为随机搜索。但考虑到跳出局部最优解的重要性，因此我们设定变异的概率为 20%。

四、实验结果。

这里以算例 DJ38 与 QA194 为例，展示遗传算法的优异效果，其他算例我们将罗列最短距离与时间的关系表格、gap 值，完整的实验结果展示在算例输出文档中。

4.1 主要实验结果展示

4.1.1 算例一：

（1） 算例信息。

算例名称：DJ38，城市：38，最短距离：6656

来源：http://www.math.uwaterloo.ca/tsp/world/countries.html#DJ

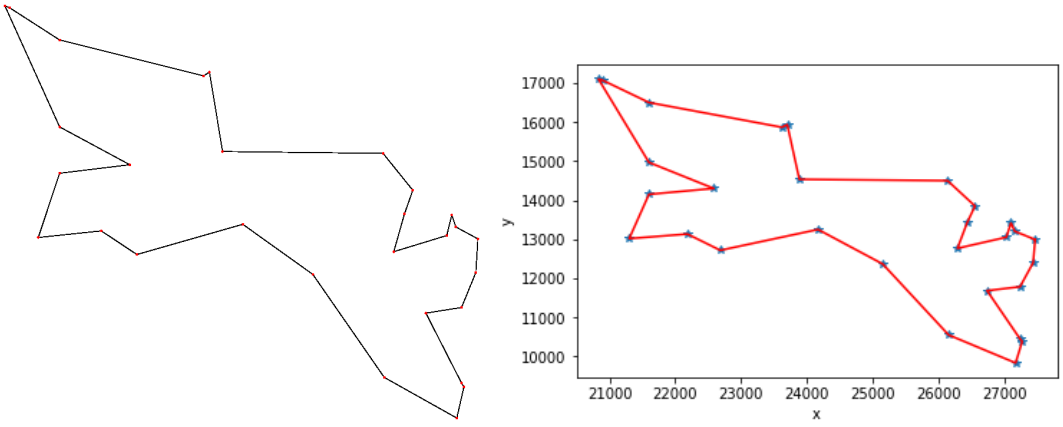
（2）最优解及可视化。

I、最优解。

29->28->20->13->9->0->1->3->2->4->5->6->7->8->11->10->18->17->16->15->12->14->19->22->25->24->21->23->27->26->30->35->33->32->37->36->34->31->29

II、最优解随迭代次数变化。

左侧图为官方给的路线图，右侧图为我们求得的最优解。

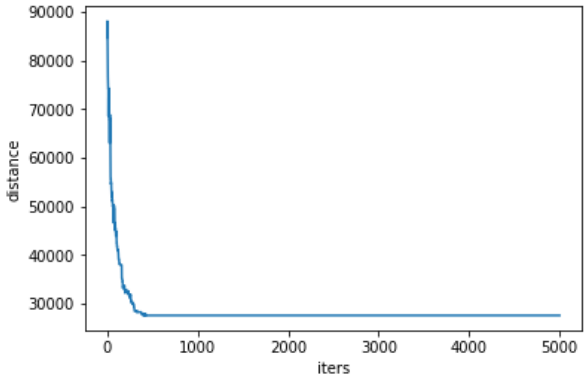


(3) 算法效率。

I、最优解随时间变化

时间	解	时间	解
10s	6710.33	3 分钟	6659.43
30s	6710.33	5 分钟	6659.43
1 分钟	6659.43	10 分钟	6659.43

II、最优解随迭代次数变化。



III、解的质量

Gap=(6659.43 - 6656) / 6656 = 0.05%

4.1.1 算例二：

（1） 算例信息

算例名称：TSPLIB , qa194，城市：194，最短距离：9352

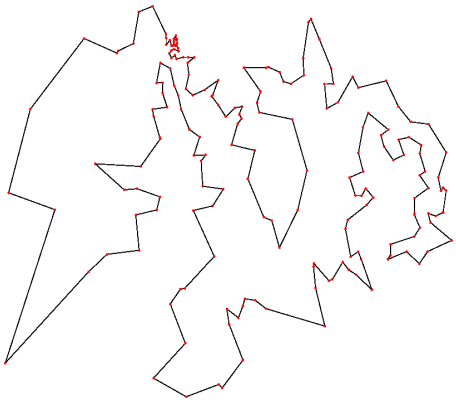
来源：http://www.math.uwaterloo.ca/tsp/world/countries.html#DJ

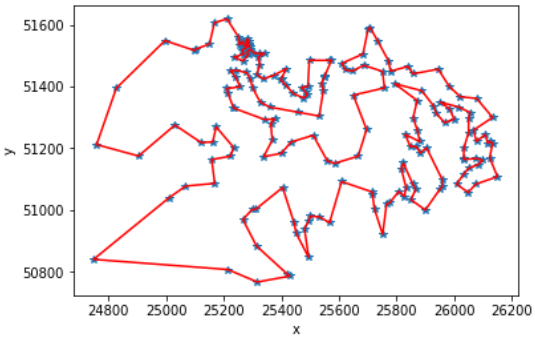
（2） 最优解及可视化。

I、最优解路线：

143->149->153->156->163->162->160->155->144->148->145->138->137->141->139->136->133->131->129->126->124->125->113->110->103->100->98->93->97->89->88->81->61->58->35->62->84->85->64->19->0->5->7->15->12->22->24->16->13->10->6->3->1->2->4->8->9->11->14->18->29->31->30->34->37->40->45->43->41->49->48->54->53->51->52->55->47->42->39->33->38->26->36->50->46->57->60->66->72->65->67->63->69->76->83->80->78->82->87->92->95->94->91->96->99->109->111->107->106->104->105->102->90->73->68->59->56->44->28->21->27->32->17->20->23->25->71->77->74->75->70->79->86->101->108->112->118->121->117->130->128->120->116->115->114->119->122->123->127->132->134->142->147->159->165->170->184->192->180->183->187->188->190->191->189->193->181->175->168->171->178->185->186->182->173->172->174->176->177->179->169->161->166->167->164->158->157->154->135->150->146->151->152->140->143

II、可视化结果：



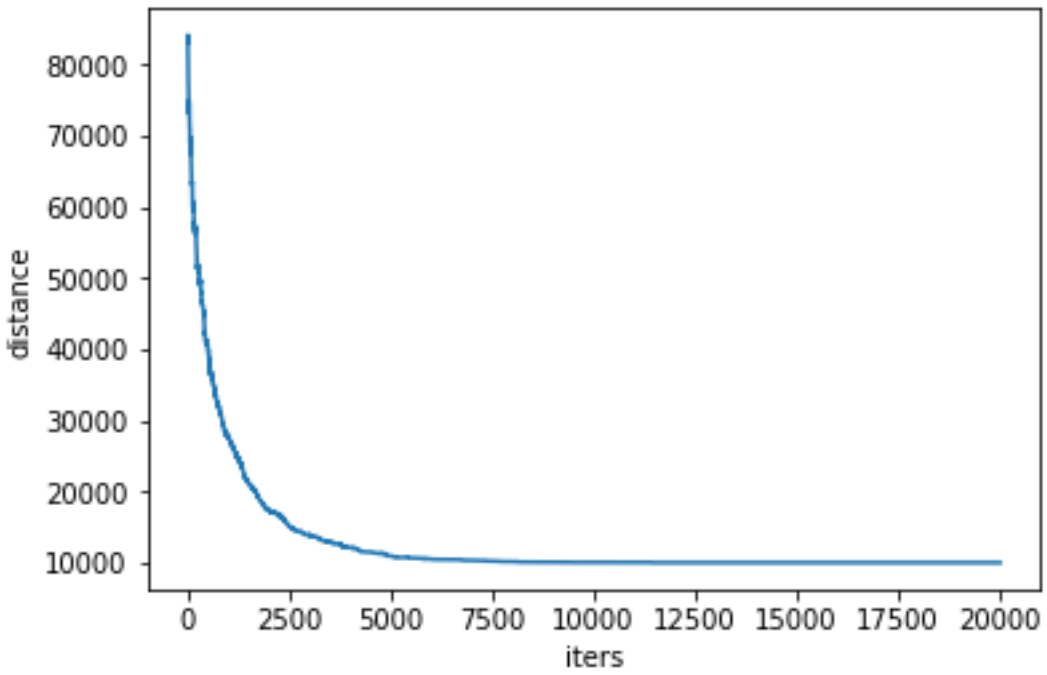


(3) 算法效率。

I、最优解随时间变化

时间	解	时间	解
10s	68889.954	3 分钟	17448.45
30s	56851.126	5 分钟	13235.822
1 分钟	34096.908	10 分钟	10087.017

II、最优解随迭代次数变化。



III、解的质量

Gap=(10087.017 - 9352) / 9352 = 7.86%

4.2 其余算例结果。

（1）xqg237

时间	解	时间	解
10s	9675.85	3 分钟	1353.26
30s	7644.74	5 分钟	1176.93
1 分钟	5973.33	10 分钟	1148.99

表：xqg237

Gap=(1148.99 - 1019) / 1019 = 7.86%

（2）xqf131

时间	解	时间	解
10s	9675.85	3 分钟	703.68
30s	7644.74	5 分钟	662.92
1 分钟	1220.94	10 分钟	634.39

表：xqf131

Gap=(634.39 - 564) / 564 = 12.48%

（3）berlin52

时间	解	时间	解
10s	9240.86	3 分钟	8179.62
30s	8297.29	5 分钟	8165.96
1 分钟	8297.29	10 分钟	8165.96

表：xqf131

Gap=(8165 - 7542) / 7542 = 8.42%

（4）wi29

时间	解	时间	解
10s	28122.50	3 分钟	27603.17
30s	28082.02	5 分钟	27603.17
1 分钟	27603.17	10 分钟	27603.17

表：xqf131

Gap=(27603.17 - 27603) / 27603 = 8.42%

4.3 平均 gap 时间。

avg_gap=6.90%。

总结：在城市数量较少时，该算法的精度较高，且收敛速度较快。当城市数量多时，算法容易收敛到局部最小值。