1. 问题和实验概述

旅行商问题（Travelling salesman problem, TSP）是组合优化领域的一个经典问题，由于其是NP难问题，因此解决此问题常常使用近似算法。本文使用了禁忌搜索算法、遗传算法和模拟退火算法进行求解，比较了各个算法参数对求解效果和计算效率造成的影响，并且比较了这几个算法的求解效果和计算效率。

本文使用的数据集是Reinelt[1]提供的TSPLIB数据集中的4组数据，节点个数分别为14、48、101和280（因为硬件性能限制，本文没有使用过于庞大的数据）。代码全部在Windows 10操作系统下基于python语言进行实现。由于这些都是随机化算法，为了保证实验可以复现，我事先固定了随机数种子。

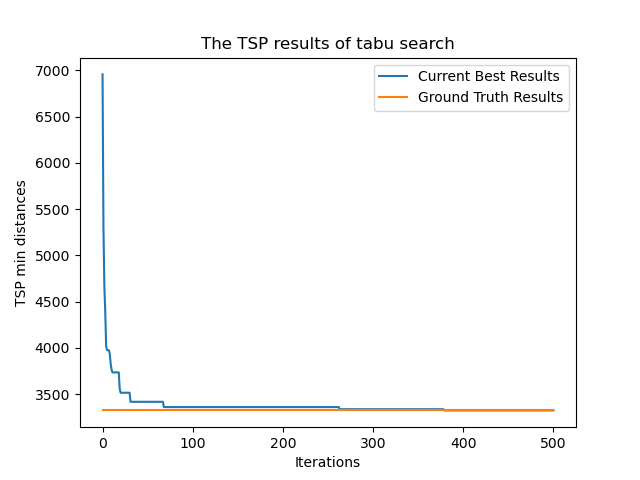
1. 禁忌搜索算法

2.1 算法介绍和实现方法

禁忌搜索算法是Glover[2]在1986年提出的经典算法。这一算法基于局部搜索算法，引入了禁忌技术以防止重复前面搜索的结果，同时引入了特赦机制来保证能够搜索到较好的解。我的实现方式如下：

1. 初始化阶段，设节点个数为N，随机生成一个访问序列作为初始解和初始最优解，初始化禁忌表为一个最大长度为M的空队列。禁忌表采用双向禁忌机制，如果禁忌（i, j）的交换，则（j, i）的交换也会被同时禁忌。
2. 迭代阶段，使用2-opt方法随机选择邻域中的K个状态，将其按照边权之和从小到大排序。
3. 看这个状态序列的第一个元素是否比当前最优值更优。如果更优，更新当前的最优解和最优值为这一状态，跳转到4，否则跳转到6。
4. 判断这个状态序列的第一个元素对应的交换是否在禁忌表里，如果不在，跳转到7。否则跳转到5。
5. 进行特赦：将这个状态序列的第一个元素对应的交换从禁忌队列中取出，加入禁忌队列头部。如果队列此时长度溢出，则弹出队列尾部元素。之后跳转到8。
6. 遍历这个状态序列，找到第一个元素使得其对应的交换不在禁忌队列中。之后跳转到7。
7. 更新禁忌队列：将找到的这个元素对应的交换加入禁忌队列头部。如果队列此时长度溢出，则弹出队列尾部元素。之后跳转到8。
8. 如果当前迭代总次数达到了最大迭代次数T，则算法终止。否则跳转到2。

在burma14数据中，我设置最大迭代次数T=500，选取状态序列长度K=20，禁忌队列最大长度M=5，成功达到了数据集报告的最优值3323，运算时间为0.27s。迭代结果变化如图1所示。



可以看出，在迭代的初期，算法得到的最优解会迅速改进。在迭代一段时间之后，算法得到的最优解会暂时保持不变，然后在某次迭代突然下降，图像呈阶梯状。这也符合禁忌搜索算法的特点：先大幅度持续改进结果，然后陷入局部最优解，再通过随机化和禁忌、特赦机制跳出局部最优解。

* 1. 参数对算法效果、性能的影响

我采用att48数据进行实验，其节点个数N=48，迭代次数T=10000，其理论最优解为10628。

首先，我固定禁忌队列长度M=10，比较不同状态序列长度K对算法造成的影响，结果如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 状态序列长度K | 10 | 20 | 50 | 100 | 200 | 500 |
| 最优解 | 16576 | 13450 | 11523 | **10972** | 11103 | 12390 |
| 耗时 | 6.02s | 9.11s | 17.35s | 32.47s | 61.95s | 142.75s |

可以看出，状态序列长度适中才能让算法效果最好：状态序列长度过小的话，由于每次搜索的范围有限，算法不容易跳出局部最优解，不容易得到好的效果。当状态序列长度过大的时候，也并不会对跳出局部最优解有很大的帮助，还会大大增加时间开销，因为求状态序列并排序的复杂度是O(KN+KlogK）。

之后，我固定状态序列长度K=100，比较不同禁忌队列长度M对算法造成的影响，结果如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 禁忌队列长度M | 3 | 5 | 10 | 20 | 50 | 100 |
| 最优解 | 11163 | 11173 | **10972** | 11062 | 11509 | 12100 |
| 耗时 | 32.45s | 32.58s | 32.47s | 32.58s | 32.87s | 32.51s |

可以看出，禁忌队列长度适中才能让算法效果最好。当禁忌队列长度过小的时候，被禁忌的搜索项很容易就会出来，让算法难以吸取之前搜索的教训，造成效果不佳。而禁忌队列长度过大的时候，被禁忌的搜索项很难出禁忌队列，导致难以跳出局部最优解。而由于算法的主要时间开销来自求状态序列和排序，因此不同的禁忌队列长度对算法效率影响不大。

2.3 结果汇总

在burma14、att48、eil101、a280四组数据下，我调整了算法的参数，得到的最优结果和对应参数如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数\数据集 | burma14 | att48 | eil101 | a280 |
| 理论最优解 | 3323 | 10628 | 629.00 | 2579.00 |
| 算法得到的最好解 | 3323 | 10972 | 722.00 | 3874.64 |
| 性能比 | 1.00 | 1.03 | 1.15 | 1.50 |
| 耗时 | 0.27s | 32.47s | 272.20s | 2561.62s |
| 迭代次数T | 500 | 10000 | 10000 | 10000 |
| 状态序列长度K | 20 | 100 | 500 | 2000 |
| 禁忌队列长度M | 5 | 10 | 5 | 5 |

可以看出，较好的状态序列长度随着问题规模的扩大在不断增加，而较好的禁忌队列长度则没有明显变化，都是5到10之间。在问题规模较小时，算法可以轻易得到理论最优解。而随着问题规模的扩大，一万次迭代也并不能得到理论最优解，而同等迭代次数下，算法的耗时在不断增加，因为算法每次迭代都需要O(KN+KlogK)的时间来求解状态序列和排序，并且随着问题规模N的不断变大，较好的状态序列长度也在不断变大。而随着问题规模的扩大，算法的性能比也在不断变差，因为需要更多的时间来跳出局部最优解。

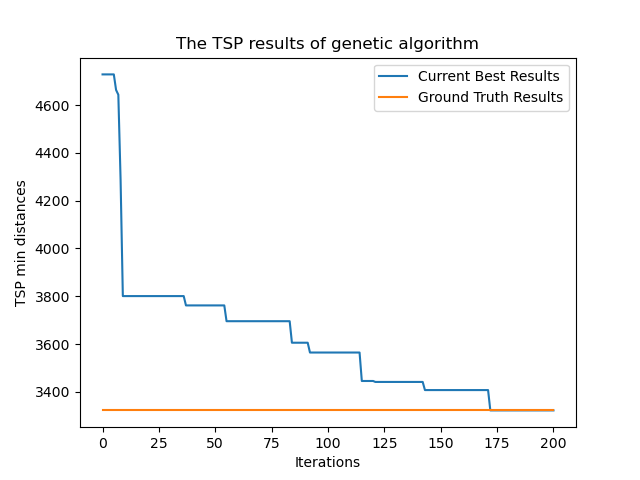
1. 遗传算法

3.1 算法介绍和实现方法

遗传算法是Holland[3]在1970年代提出的算法。此算法借用了生物学中种群交配、基因变异、自然选择和进化的内容，通过交叉、变异和选择来不断改进种群中的实例，进而收敛到全局最优解。我的实现方法如下：

1. 初始化阶段，设TSP问题的节点个数为N，初始种群大小为M，我随机初始化M个城市访问序列作为初始种群，求出其总距离，并且记录里面距离最短的访问序列作为初始最优解。
2. 进入遗传算法主循环，总共循环T次。为了保证遗传算法收敛到全局最优解，我们对几个步骤的执行顺序做了调整：首先在种群中进行交叉、然后进行变异、然后对种群的所有访问序列按照总距离从小到大排序更新最优解、最后求适应度函数进行种群选择。
3. 交叉阶段：我将1到M这M个访问序列随机分成M/2组，每组2个序列。对于每组数据，按照交叉概率pc进行交叉。我的交叉方法是两点交叉法，也就是选择两个端点，对组中两个序列在这两个端点之中的子序列进行交叉。交叉之后两个序列在端点两边的部分可能会产生重复，因此还需要进行去重处理，即记录下重复的位置，使交叉双方重复的节点进行交换。
4. 变异阶段：对于种群中的每个访问序列，我们都按照变异概率pm进行变异。我的变异方法是选择两个端点，对两个端点之中的子序列进行取反。
5. 更新最优解阶段：对于现在的种群，我们将其按照总距离由小到大排序，并且更新最优解和最优值。
6. 选择阶段：首先，需要设定适应度函数。我设计了两种不同的适应度函数：一种是距离适应度函数，即总距离倒数的平方；一种是排序适应度函数，即将种群按照总距离由大到小排序，总距离最大的适应度为1，第二大的适应度为2，以此类推。之后，我们采用轮盘赌算法进行选择，保留M个序列。

在burma14数据中，我设置最大迭代次数T=200，初始种群大小M=50，交叉概率pc=1，变异概率pm=0.05，适应度函数为距离适应度函数，成功达到了数据集报告的最优值3323，运算时间为0.40s。迭代结果变化如图2所示。



可以看出，在迭代的初期，算法得到的最优解会迅速改进。在迭代一段时间之后，算法得到的最优解会暂时保持不变，然后在某次迭代突然下降，图像呈阶梯状。这也符合遗传算法的特点：首先迅速优化种群，然后通过随机化的交叉、变异机制寻求跳出局部最优解。

* 1. 参数对算法效果、性能的影响

我采用att48数据进行实验，其节点个数N=48，迭代次数T=10000，其理论最优解为10628。

首先，我分析初始种群大小M对结果的影响，固定交叉概率pc=1, 变异概率pm=0.2， 适应度函数为距离适应度函数：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 初始种群大小M | 10 | 20 | 50 | 100 | 200 | 500 |
| 最优解 | 12807 | 10897 | 11045 | **10771** | 11156 | 18397 |
| 耗时 | 10.04s | 14.15s | 32.10s | 92.02s | 178.53s | 549.03s |

可以看出，初始种群大小适中才能让效果最好。初始种群太小的话，可供选择、交叉和变异的个体有限，不一定能搜索到最优解，交叉和变异的范围也不够大，难以跳出局部最优解。初始种群过大的话，会让交叉、变异过于频繁，反而不利于搜索到最优解，而且运算速度也大大减缓。

其次，我分析交叉概率pc和变异概率pm对结果的影响。

我固定初始种群大小M=100，适应度函数为距离适应度函数，变异概率pm=0.2。不同交叉概率pc对应的结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 交叉概率pc | 0 | 0.5 | 0.7 | 1 |
| 最优解 | 35787 | 13207 | 11082 | **10771** |
| 耗时 | 20.02s | 56.47s | 66.73s | 92.02s |

我固定初始种群大小M=100，适应度函数为距离适应度函数，交叉概率pc=1。不同变异概率pm对应的结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变异概率pc | 0 | 0.05 | 0.1 | 0.2 |
| 最优解 | 21804 | 11154 | 10905 | **10771** |
| 耗时 | 70.37s | 77.50s | 82.86s | 92.02s |

首先，可以看出，交叉和变异都必不可少，没了任何一个机制，整体的效果都会大打折扣。其次，如果选择的初始种群大小和适应度大小足够合适，交叉概率和变异概率完全可以多多益善（变异概率不能超过0.2，否则会退化成随机化搜索算法），因为这样可以最大限度地增加算法的随机性，提高跳出局部最优解的可能。

最终，我分析适应度函数对结果的影响。我固定初始种群大小M=100，交叉概率pc=1，变异概率pm=0.2，不同适应度函数对应的结果如下：

|  |  |  |
| --- | --- | --- |
| 适应度函数 | 距离 | 排序 |
| 最优解 | **10771** | **10771** |
| 耗时 | 92.02s | **88.86s** |

可以看出，当种群大小、交叉概率、变异概率都选取的较为合适的时候，两个适应度函数差别并不大，两者得到的最优解相同，而排序适应度函数耗时更少。对于不同的数据，两者各有优劣：距离适应度函数对于距离较为敏感，对于距离范围和差异较大的数据更能得到较好的结果；排序适应度函数对于距离没那么敏感，对于距离范围和差异较小的数据更能得到较好的结果。

3.3 结果汇总

在burma14、att48、eil101、a280四组数据下，我调整了算法的参数，得到的最优结果和对应参数如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数\数据集 | burma14 | att48 | eil101 | a280 |
| 理论最优解 | 3323 | 10628 | 629.00 | 2579.00 |
| 算法得到的最好解 | 3323 | 10771 |  |  |
| 性能比 | 1.00 | 1.01 |  |  |
| 耗时 | 0.40s | 92.02s |  |  |
| 迭代次数T | 200 | 10000 |  |  |
| 初始种群大小M | 50 | 100 |  |  |
| 交叉概率pc | 1 | 1 |  |  |
| 变异概率pm | 0.05 | 0.2 |  |  |
| 适应度函数 | 距离 | 距离 |  |  |

[2] Glover, F. (1986) “Future Paths for Integer Programming and Links to Artificial Intelligence,” Computers and Operations Research, Vol. 13, pp. 533-549.

[3] Holland, J. H. (1975). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press.