局部搜索效率对比实验报告

王资 18214668 计算机技术

1.基本思想

对比基本 n-queens 问题的基本搜索方法（深度优先搜索）、一般局部搜索（爬山算法）、避免陷入局部最优解的局部搜索算法（陷入局部最优时，重新随机初始化），三种算法的程序运行效率。

2.代码中的数据结构

int n，代表 n 后问题的规模；int* x，代表棋盘；bool* conf 标识当前棋盘中存在冲突的皇后；int threshold 对于爬山算法，设置一个在陷入局部最优时可以退出循环的阈值。

3.程序流程

a)局部搜索算法

对当前的棋盘，交换任意二个皇后的位置，记录下交换后的冲突数，交换过所有可以两两交换的皇后之后，选择冲突数最少的交换方式进行交换，进行下一轮循环。

b)避免陷入局部最优的在局部搜索算法

对当前的棋盘，交换任意二个皇后的位置，计算交换后的冲突数，若比当前的冲突数少，则立刻接受这种交换；若陷入局部最优，即进行过所有可能的两两交换，冲突数仍不能下降，则重新随机打乱棋盘。

4.调试过程出现的问题和解决方法

一般的局部搜索算法对于恰当的初始棋盘要求较高，否则会陷入局部最优，并且在循环中不能退出，程序陷入死循环。因此要加入一个阈值，统计陷入局部最优的次数，若超过阈值则直接跳出循环并将本次搜索标识为失败的搜索。如果陷入一次局部最优就跳出，则失败率会更高，因此修改为可以进入冲突数与当前冲突数一样的摆法，但统计进入次数，超过阈值才跳出。

5.运行结果

横向对比：

```
/home/wz/CLionProjects/n-queens/cmake-build-debug/n_queens
106 62 127 85 54 46 61 134 115 4 22 63 84 130 123 69 100 10 71 29 148 58 27 146 68 15 17 53 1 44 34 50 12 116 3 36 131 108 56 97 65 24 28 121 75 30 143 95
150-queens in random local search takes 2s
0 2 4 1 3 8 10 12 14 16 18 22 24 26 23 25 5 9 6 15 7 11 13 20 17 19 21
27-queens in standard search takes 2s
23 39 33 3 58 60 47 50 10 35 37 30 7 18 20 14 9 62 19 17 0 45 34 31 28 42 4 6 44 40 52 8 46 22 55 11 27 5 32 25 21 54 16 53 29 63 61 12 1 56 59 24 48 41 3
64-queens in local search takes 1s

Process finished with exit code 0
```

在 1-2 秒内

深度优先搜索可以找到 27 后的解

局部搜索可以找到 64 后的解（如果不是失败的情况）

带随机的局部搜索 150 后的解

注：大部分情况一般的局部搜索是搜索失败的。

纵向对比：

```
/home/wz/CLionProjects/n-queens/cmake-build-debug/n_queens
0 68 57 62 103 27 119 94 106 78 63 86 142 149 31 79 105 40 53 24 136 77 33 70 129 58 44 55 60 130 7 35 29 47 134 128 28 132 80 93 12 6 108 3 141 54 114 1 42 85
150-queens in random local search takes 1s
84 57 123 47 31 91 128 7 105 109 127 106 115 92 81 26 135 122 150 3 126 79 121 59 64 169 12 42 165 101 161 16 46 111 41 163 68 62 90 164 130 174 70 80 37 74 14
175-queens in random local search takes 3s
118 87 134 128 64 151 115 197 39 158 31 167 90 133 30 148 56 198 62 48 83 73 98 170 46 49 166 96 7 108 141 102 175 146 93 135 89 40 165 113 125 179 157 66 72 1
200-queens in random local search takes 4s
193 140 103 106 51 121 78 91 145 166 148 54 125 44 66 151 75 172 81 126 48 142 219 129 2 4 16 32 150 47 70 40 197 118 88 111 168 41 102 201 216 158 213 134 63
225-queens in random local search takes 9s
224 119 192 72 200 140 146 74 105 132 4 137 197 233 191 145 11 154 177 111 167 169 143 215 38 49 171 198 94 92 30 219 59 75 69 164 96 10 121 101 114 123 26 54
250-queens in random local search takes 16s
227 160 147 127 211 155 184 83 121 86 177 100 164 235 84 225 104 58 23 167 153 137 141 122 230 259 197 117 270 10 158 9 162 49 106 239 210 224 36 57 138 148 89
275-queens in random local search takes 23s
82 116 94 75 240 126 213 8 55 71 150 194 168 125 42 84 200 22 286 196 80 63 112 142 133 252 130 199 214 122 2 169 66 36 245 136 0 83 141 212 268 146 101 222 15
300-queens in random local search takes 62s
179 108 206 228 237 158 101 62 225 37 220 64 56 207 221 146 284 19 216 139 102 78 89 219 128 214 8 113 265 138 34 246 295 180 125 257 318 7 279 85 51 291 96 15
325-queens in random local search takes 48s
178 197 158 108 306 80 226 121 229 308 340 46 153 264 335 177 147 54 161 30 270 72 238 176 156 194 190 24 131 73 142 301 155 115 295 239 289 76 17 22 186 183 3
350-queens in random local search takes 65s
```

带随机的局部搜索在一分钟内，最多可以找到 300～350 后的解。

6.总结

本次实验旨在实现、对比局部搜索算法与标准全局搜索算法之间的效率，由于局部搜索算法是更有目标的，比起标准全局搜索算法的剪枝，前者能够剪掉更多的分支，但随之而来的是对于某些非线性的问题，可能会陷入局部最优的情况，需要一些改进算法来避免这种情况，本次的 n 后问题就使用了重新初始化的方法来避免陷入局部最优，由于随机初始化是带随机性的，因此若某次随机出来的初始化棋盘可以用局部搜索找到最优解的话，就能得到最优解，否则会重新下一次随机初始化，所以效率的浮动是比较大的，需要大量测试，得到平均值作为结果。

使用这种随机初始化的局部搜索算法，得到了在 1s 比全局搜索，即深度优先搜索算法效率提高 5 倍的改进（对于 n 更大的情况，改进更多，比如深度优先搜索找到 30 后需要几分钟，但相同的时间，随机初始化的局部搜索可以找到大于 400 后的解）。

7.存在问题和改进设想

虽然这种随机初始化的局部搜索算法既比全局搜索效率有了很大的提高，又能避免陷入局部最优解的情况，但与现有的高效搜索效率相比还是较低的，比较突出的问题是解 1000 后问题需要约一小时。可能的问题是计算冲突数需要的时间消耗较大，是 O(n^2)量级的，而每次皇后位置的交换只交换了 2 个，因此可能可以只关注这 2 个变化了的值，使得冲突数计算可以下降到 O(n)量级，这样可能可以进一步改进算法。使用更优的局部搜索算法如模拟退火等可能也能进一步优化搜索效率。