浅谈估价函数在信息学竞赛中的应用

浙江省绍兴一中　周而进

目录

摘要

估价函数在状态空间庞大，状态转移费时的一类逐步求精问题求解中有重要的应用。本文从 3 个方面，浅谈了估价函数在信息学竞赛中的应用，讲解了估价函数的选取、评估、应用，解决原本难以下手的问题。

关键字

估价函数 逐步求精 搜索 动态规划 较优性 平衡性

前言

估价函数是将状态在未来求解中的优劣映射到一个具体数值的一种评估。举一个简单的例子，在一个国际象棋博弈中，一个刚学会下棋的小孩总会被告知：1 个兵 1 分，一个象或马 3 分，一个车 5 分，皇后 9 分，有了这么一个标准，自然而然会得到一个最简单的估价，以下一步能吃掉对方的子的分值作为下一步的估价函数，如果将这个估价函数应用到程序中，那原本只会暴力搜索的程序就具备了初级棋手的实力了。

信息学竞赛中题目往往变化多端，在无法找到最优方法解决问题的时候，通过合理运用估价函数，时常能达到出人意料的效果，事半功倍。

1 估价函数在搜索中的应用

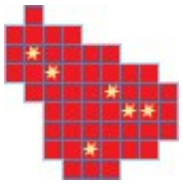
1.1 在一类最优性问题中的应用

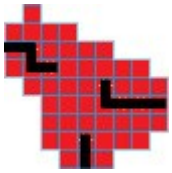
在信息学竞赛中，我们往往会碰到一类最优性问题的求解，而问题本身很难找到一个有效的多项式解法，于是只能采取搜索策略。但这类问题往往状态空间庞大，直接暴力搜索肯定效率低下。如果我们运用估价函数，就能及时对搜索进行剪枝，从而大大提高程序的效率。

1.1.1 Mines For Diamonds[1]

1.1.1.1 问题描述

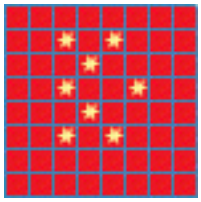
给定一个 n*m 钻石矿区，你需要挖一些地道将所有的钻石挖出来：
一条地道是一条从边境出发的折线，途中覆盖的钻石都算采到，两条折线不能相交，但可以相邻，求最小的地道总长度
保证 n，m≤11，钻石个数≤10





1.1.1.2 分析

首先，因为这个题范围比较小，也没有别的什么多项式方法，所以就考虑搜索。直接写暴力自然是 TLE 的，所以需要剪枝。来看一个数据：



来看看这个暴力的效果：

深度	1	2	3	4	5	6
生成节点数	9	41	145	515	1776	6120

1 UVa Online Judge 10605 Mines For Diamonds

深度	7	8	9	10	11	12
生成节点数	20511	67935	220273	702437	2191682	6705638
深度	13	14	15	16	17	
生成节点数	20066463	58721500	167773977	467536220	120043037	

可见到深度较深的时候，扩展的节点数疯狂上涨，所以如果我们能在前期就剪去一部分节点，那将大大提高程序的效率。自然而然的，就想到了通过估价函数进行剪枝：如果 $g+h \geqslant ans$，那就没有必要继续搜索下去，这里 g 表示到达目前状态的花费，而 h 就是对当前局面转变成目标状态的一个乐观估计，也就是需要保证 $h \leqslant h^*$，h^*是真实的需要的步数。

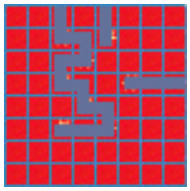
于是，如何设计估价函数 h 成了主要问题，重新审题发现，这里钻石的个数不是很多，而本问题的一个主要限制在于两条地道不能相交，如果我们忽略的这个条件，那原问题就转化为了：给定一个点集，需要你将他们划分成一些子集，每个子集的点通过一条链相连，要求总和最小。而这个问题是可以通过一个二次 dp 完成的，令 $opt[s]$表示点集 s 的最优解，则有：

$$opt[s]=\min\{opt[s-t]+w[t],\ t \subset s\}$$

这里 $w[t]$表示将点集 t 用一条链连起来的最小费用：

$$w[t]=\min\{w[t-k]+dist[t-k][k],\ k \in t\}$$

如果将 $opt[s]$作为估价函数，首先他满足 $h \leqslant h^*$，并且在大多数时候，这也是非常接近最有解的：



对于这个数据，现在这个加了估价函数的程序，跑出的结果就是：

深度	1	2	3	4	5	6
生成节点数	0	0	0	0	0	0
深度	7	8	9	10	11	12

生成节点数	0	0	0	0	0	0
深度	13	14	15	16	17	
生成节点数	0	0	2112	43342	16733	

优化效果不言而喻，在 UVa 上也能以 0.1s 以内 AC。

1.1.2 11163 Jaguar King[2]

1.1.2.1 问题描述

有 n 只豹子要排队 ，位置为 $1..n(n\%4=0)$，，其中编号为 1 的为豹王，只有豹王可以和别的豹子交换位置，豹王跳跃有限制，设当前豹王位置为 i：

如果 $(i\%4=1)$ 那么他可以跳到$(i+1)$，$(i+3)$，$(i-4)$，$(i+4)$
如果 $(i\%4=2)$ 那么他可以跳到$(i+1)$，$(i-1)$，$(i-4)$，$(i+4)$
如果 $(i\%4=3)$ 那么他可以跳到$(i+1)$，$(i-1)$，$(i-4)$，$(i+4)$
如果 $(i\%4=0)$ 那么他可以跳到$(i-3)$，$(i-1)$，$(i-4)$，$(i+4)$
求豹王最少要跳几次才能排完序，$n\leqslant 40$

1.1.2.2 分析

这个问题很眼熟，是一个排序，又只有一个位置能移动，很像 8 数码问题这个经典模型，但是 8 数码里空格是可以随意移动的，而这里王的移动却有了限制。其实，如果我们将豹子按照 $X*4$ 的格式排队，就会有新的发现：

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

……

就以第二行为例，5 可以跳跃到的是$(5+1)$，$(5+3)$，$(5-4)$，$(5+4)$，刚好就是 5 的相邻 4 格：（6，8，1，9），同样 6 能跳到的就是$(5，7，2，10)$，7 能跳到的就是$(6，8，3，11)$，8 能跳到的就是$(7，5，4，12)$。其实每个点能跳到的就是相邻的 4 个点，所以其实也就是和 8 数码差不多。

于是，和 8 数码一样，我们就得到了一个估价函数 $E(s)$：

$$E(s)=\sum dist[i][pos[i]],\ i>1$$

$dist[i][j]$表示点 i 和点 j 在 $X*4$ 的局面下的曼哈顿距离。

2 UVa Online Judge 11163 Jaguar King

写成代码就是：

```
x=abs((i-1)/4-(j-1)/4);
y=abs((i-1)%4-(j-1)%4);
if (y==3) y=1;
dist[i][j]=x+y;
```

这样在迭代深搜的过程中加入这个估价，就能很快通过 UVa 上的数据。

1.1.2.3 一些拓展

这个问题有一个拓展，这里王的限制是有规则的，如果题目拓展成，给定一个点集 V，对于每个点 V_i，分别有一个可达点集 $(V_{i1}, V_{i2}, \ldots, V_{ik})$，花费分别是 $(W_{i1}, W_{i2}, \ldots, W_{ik})$，求将点集排序的最小代价。这里不再存在曼哈顿关系的估价了，不过我们可以找到一个更一般的方法，通过松弛 (relaxation)进行估价。对于点 V_i 的可达点集，我们建立边 $(E_{i1}, E_{i2}, \ldots, E_{ik})$，权值是 $(W_{i1}, W_{i2}, \ldots, W_{ik})$，那么估价

$$E(s)=\sum ssp[i][pos[i]],\ i>1$$

这里 $ssp[i][j]$ 表示在图 $G=(V, E)$ 中的 i，j 的最短路。

1.1.3 小结

对于在搜索中求最优解的一类问题，盲目搜索往往会遍历所有情况，复杂度自然是难以想象，及时对局面进行评估成为了提高程序效率的重要因素。一个完美的估价最好就等于真实的花费，不过实际很难找到这样的“神谕”，所以我们往往采取忽略题目中的某一个要求的方法，将题目转化成一个较容易的问题，从而求出原问题的一个解的上限，而对于新问题的求解往往能用贪心，动态规划等高效的方法，从而达到大量减少无用状态搜索的作用。

1.2 在一类可行性问题中的应用

有一类搜索问题，题目并不需要求一个最优解，仅需要求一个可行解，这类问题的状态空间往往更加庞大，盲目搜索自然效率低下，而且一般的最优性剪枝直接套用可能导致效率大大降低，得不偿失。解决这类问题的一个有效的方法就是通过估价函数。下文从一个经典的例子讲解了估价函数在解决这一类问题中发挥的作用。

1.2.1 Sokoban[3]

1.2.1.1 问题描述

给定一个 $n*m$ 的推箱子的局面，求一个不超过 10000 步的解，题目保证有解，且 n，$m \leqslant 8$。

为了方便接下来叙述，这里按题目的约定：

- `#` 表示墙
- 。 表示一个目标点
- `@` 初始人的位置
- `+` 初始人的位置且这个位置是一个目标点
- `$` 箱子的位置
- `*` 箱子的位置且这个位置是一个目标点

```
########
#@  $ .#
########
```

```
  ######
##    .#
#@  ###
#   * #
#   $ #
#     #
#######
```

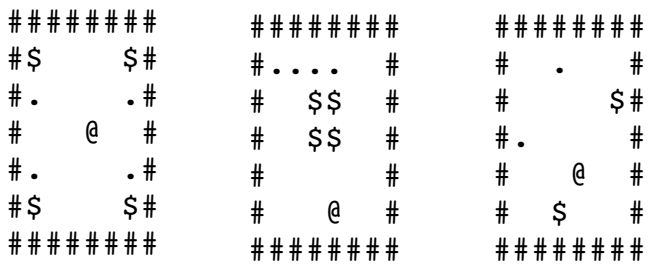
1.2.1.2 分析

这是 Ural 上的一个经典难题，提交次数 2000 多次，AC 人数目前只有 6 个，通过率不到 1% !

首先，先分析一下这个题的数据范围，题目保证 n，$m \leqslant 8$，去掉外边的一

3 Timus Online Judge 1589

圈墙，所以实际面积最多只有 6*6，也就是说箱子的个数最多也就 35。



接着是确定搜索的方法，因为是多个箱子，所以为了方便写，只好一步推一个箱子，直到所有的箱子都归位。但是这样很容易出现的一个局面就是死锁。比如一个箱子推到了墙角，4 个箱子相互靠在了一起，一个箱子贴在了没有目标点的一边墙上等等。

当然死锁远远不止这些，有些人一眼就能看出来，但电脑却不一定了，而且频繁地判断死锁会降低程序的效率。既然正想比较麻烦，不如反向思考，将推箱子变成拉箱子，这样就不会出现上诉的种种死锁了，一步就解决的问题。于是通过将推箱子变成拉箱子，第一个程序就出来了，同时，为了加速，我用上了 hash，我用他跑了一下几组数据，基本没出解，不过样例过了。而且关键部分还没出来呢。

这里我们定义一个局面 $S(P, Q, pos)$，P 为箱子的集合，Q 为目标点的集合，pos 为当前人的位置。因为是写迭代加深，所以自然而然的想到了估价。通过迭代加深，我们枚举答案上限 MAX，设到当前局面的代价是 g，如果 $g+h>MAX$，便可以剪枝，从而大大提高效率。

对于估价函数，一个较显然的结论是，点$(x1, y1)$到点$(x2, y2)$的花费至少是他们的曼哈顿距离$|x1-x2|+|y1-y2|$，那么我们设计估价 $h(s)$为：

$$h(s)=\sum \min\{|Xi-Xj|+|Yi-Yj|,\ j\in Q\},\ i\in P$$

这样的估价效果如何呢？这里我选了 33 组数据进行测试[4]：

test	生成节点	test	生成节点
0*	3 3 3 597 5360 23115...	17	3 44 5935 5703
1	4 4 4 185 34	18	3 176 421 164
2	3 43 69	19	8 233 56

4 打*表示最终没有跑出解来

3*	3 3 3 597 5360 23115 20718...	20	1 68 579 810 741
4	4 4 168 52	21*	7 12 10828 78396...
5*	4 4 1003 10039 32094 51369 ...	22	5 98 2628 1557
6	5 33 993	23	9 100 1306 96
7	6 39 2712 11309 2728	24	3 16 805 587
8	4 7 454 894 1038 1385 1487 402	25*	6 62 1006 2184 1970 1756 ...
9	1 42 83	26*	2 2 27300 179942 ...
10*	4 4 2275 40402 120366...	27*	4 4 1621 10704 20644 29435...
11	3 3 20	28	5 71 1955 3394 5852 3827
12	5 92 120	29	5 5 2071 10659
13	6 147	30*	4 4 354 12355 44301 105225 ...
14	3 3 214 945 615	31	1 1 676
15	3 59 65 53	32*	1 1 ...
16*	5 5 313 9927 64003 49533 ...		

这里是以 10 为单位步长进行迭代深搜的，这里生成的节点表示的是推箱子的节点数，不是最终的答案的步数(答案的步数绝大多数为人移动的路径)。

由表可见，由于顶层节点控制地不是很好，导致了后期节点疯狂上涨，所以有必要优化这个估价。

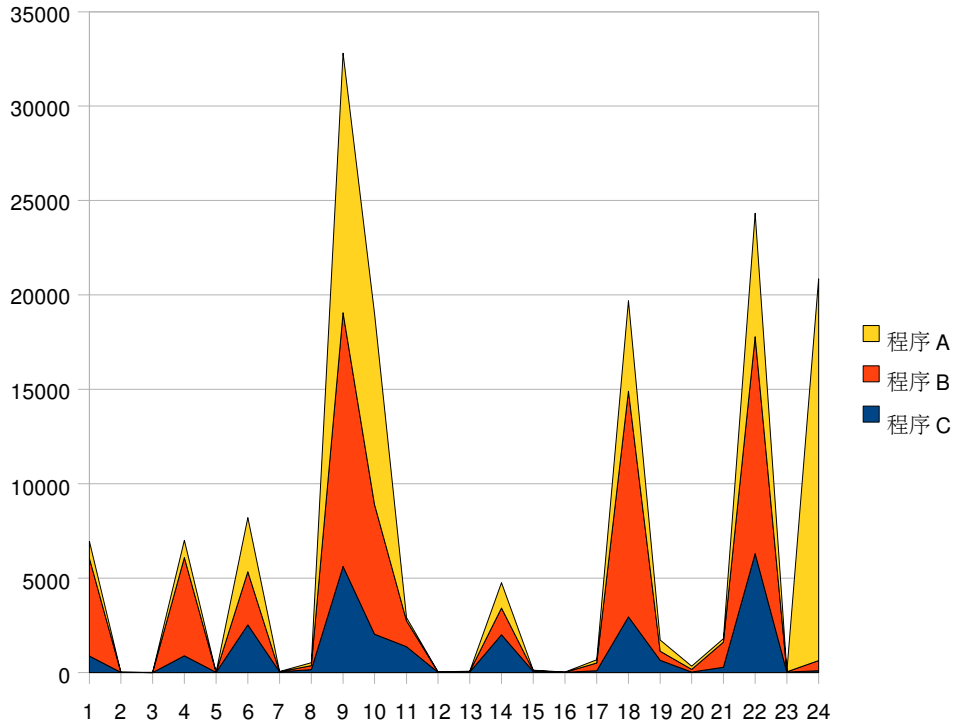
这里是点对点的关系，如果直接使用 min，的确会造成比较离谱的估价，既然如此，就不妨试试 KM，虽然 KM 复杂度是 $O(n^3)$ 的，但这里首先 n 比较小，最多 35，其次这里如果能在早期剪掉比较多的点，对后期会有很大的帮助，或许还是很有效的，结果如下：

test	0*	1	2	3*	4	5	6	7	8	9	10*
节点	53838	34	69	53838	35	20421	514	4788	598	78	226089
test	11	12	13	14	15	16*	17	18	19	20	21
节点	20	135	132	4511	59	67341	15502	67	223	1023	5556
test	22	23	24	25	26*	27	28	29	30*	31	32
节点	1305	94	755	2828	70160	8054	4463	74378	163117	542	23859

的确又有了一些新的点跑出来，但是 $O(N^3)$ 的 KM 如果每次都做一次的话，

这个估价的代价就太大了，仔细观察就能发现我们能对这个 KM 进行优化。对于一个已经做好的最优匹配 M，由于一次推箱子只改变了一个箱子的位置，所以没有必要重新做一边 KM，新匹配 M' 于原匹配 M 相比最多只有 2 对匹配发生变化，所以只要枚举另一对匹配就可以了，这样我们就把除了初始那一次以外的 KM 都优化到了 $O(N)$。

这样看，估价函数方面的优化似乎已经是山穷水尽了，其实不然，这里我们只是用估价函数进行了最优性剪枝，但估价函数本身是对一个局面的估价，通过估价函数，我们就可以比较两个局面的优劣，也就是说如果在每一个节点扩展其子节点之前，先对节点按估价函数排序，越有前途的子节点越先搜索，加了这个优化，所有的测试点都跑出来了，并且顺利通过了 Ural 上的所有数据。



3 个程序的时间效率对比如上（其中横轴表示数据点数，纵轴表示时间，以 ms 为单位），程序 A 以曼哈顿距离取最小值为估价，程序 B 以曼哈顿距离配合 KM 作为估价，程序 C 是在程序 B 基础上带节点排序。

这里有一个问题，这个估价函数 $h(s)$ 并不是单调的，也就是说我不能保

证每次找到一个点的时候就一定是最优（最短）路径，而又因为是迭代深搜，限制了深度，所以当我们第一次在较深的深度访问到了这个点，并将它 hash 了，下次如果在较浅处再次访问，就会因为 hash 的原因不再继续搜索它的子树，从而丢掉解。其实，这是没有问题的，因为在这个问题里，我们求的是一个可行解，所以加入我们在当前层因为估价的问题丢掉了解，那么在后几层中会重新找到解，但更重要的是，通过这个估价，可以大大提高程序的效率，而这是利用了题目只要求可行解的这个特殊条件。

1.2.2 小结

对于一般的可行性问题的求解，状态空间比较大，所以设计的估价函数必须要比较好的体现出状态之间的差别，这种差别不仅能及时知道当前状态是否有存在的价值，同时也能比较两个状态的优劣，从而较快地找到解。在设计估价函数的时候，有时会遇到估价函数非单调的情况，其实在这类可行性问题中大可不必计较，因为这并不会导致解找不到，而且通过此还能换来程序效率的提升，何乐而不为呢?

2 估价函数在优化一类动规方面的应用

动态规划往往以简洁高效著称，最大限度地减少重复劳动。但是如果碰到一类状态数目过多的题目，直接的动态规划就会力不从心，这时我们就可以从估价函数方面下手，优化算法，大大提高动规的效率。

2.1 USACO DEC08 GOLD Largest Fence

2.1.1 问题描述

Farmer John 有 $n(5 \leqslant n \leqslant 250)$ 个栅栏点，他需要围成一个栅栏圈，这个圈是一个凸包并且凸包上的点最多。

2.1.2 分析

首先，这个问题的一个 $O(N^4)$ 的解法是比较简单的，首先就是枚举最终答案中凸包的最左端点，把所有在他右侧的点按极角排序，然后就是用状态 $opt[i][j]$ 表示目前围成的“凸包”的末两个点是 i，j，转移就是：

$$opt[i][j]=\max\{opt[j][k]+1\},$$

Point(i)在Line(k，j)的左侧

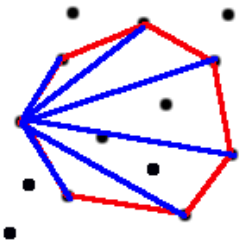
这个复杂度 USACO 官方题解说是能得到 11/20 的分数，其实写得好一点跑到 13，14 组是没问题的。但是对于大数据来说仍然是太慢了。

这个题存在 $O(N^3)$ 的解法，但是和这个短小的暴力 N^4 相比还是过于复杂，有没有既短小，又高效的解法呢?
答案就是运用估价函数。

优化一：

观察上面的方程，状态有 N^3 个，转移是 $O(N)$ 的，所以导致了复杂度变成 $O(N^4)$。如果我们能够设计出一个估价函数 $h(s)$，能及时判断当前状态是否能导出更优的解，从而减少状态，那么效率就会大大提高。

观察一个凸包后，我们可以发现凸包的一点到其他每一个点的距离是成峰形的：



也就是说，或许可以用这个性质指定估价函数，这是一个简单的非升非降的最长序列的模型，可以用

DP 简单解决，用 $up[i]$ 表示从点 i 开始的最长单峰序列，$down[i]$ 表示从点 i 开始的最长下降序列，则：

$$down[i]=\max\{down[j]+1\},\ j>i,\ dist[j]<dist[i]$$

$$up[i]=\max\{\max\{up[j]+1\},\ down[i]\},\ j>i,\ dist[j]>dist[i]$$

$$h[i][j]=up[j]$$

于是，如果对于状态 $opt[i][j]+h[i][j]\leqslant ANS$，那么说明没有必要再对这个状态进行扩展，因为一定不会导出最优解。

通过这一步优化，对于极限的 $N=250$ 的数据，运行时间从原来的 10+s 直接讲到的 2.x s，优化效果明显，但是这个题的时限是 1.2s，所以仍然不能通过所有数据。

优化二：

其实说到这，应该能联想到在 Sokoban 一题中，估价函数还起到了一个比较两个状态优劣的作用，而在这里，估价函数同样也能这么做。在同一层的转移我们可以优先处理估价高的子状态，这么做的好处就是能加速我们最优解找到的速度。当前解越优，就越加大了剪枝的力度，从而提高了程序的效率。通过这一步优化，程序对于极限数据也能小于 1s 出解，当然也就能很快通过 USACO 的所有数据了。

2.2 小结

动态规划往往以高效著称，但是状态数巨大，状态转移频繁的状况，直接用动态规划一般很难达到目的，对于这一类题目，如果应用估价函数，特别是对于记忆化搜索一类的题目，就能大大减少状态数，通过状态估价的比较，舍去估价较低的状态或者优先处理估价高的子状态，就能大大提高效率。

3 估价函数在一类较优解求解问题中的应用

竞赛中往往会碰到一类求较优解的题目。这类题目并不需要你得到最优答案，但却需要你的答案尽量优。这类问题较前面的问题往往更加灵活，也更考察选手的判断决策能力。这类问题一般自身是NPC或很难在短时间找到有效的多项式解法的问题，解决的方法很多，而通过估价函数，我们往往能得到一个简洁高效的算法，在实际应用中性价比很高。

3.1 TSP问题

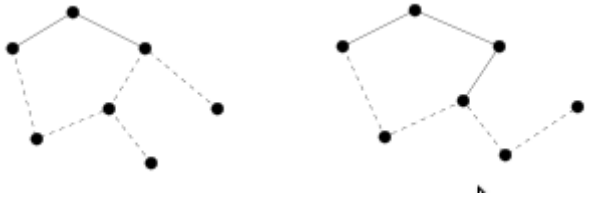
3.1.1 问题描述

本问题就是旅行商问题，给定平面上n个点，求一条回路，每个点有且仅有被遍历一次，求最短的回路。这是一个经典的NPC问题，至今尚未发现有多项式解法，对于一般图求一组最优解是非常困难的，在实际应用中往往只需要一个较优解就可以。

3.1.2 分析

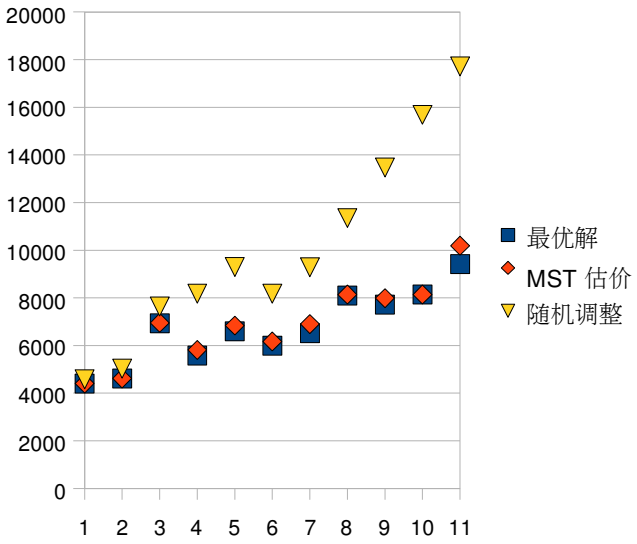
这是一个经典的模型，对于较优解，各种方法层出不穷，包括随机调整、贪心、局部动规等等。这里我要讲的就是利用搜索配合估价函数来求得不错的较优解。

对于一个局面s，一个可行的估价首先要满足$h(s)\leqslant h^*(s)$，同样是放宽限制，我们发现，题目中要求的是一条汉密尔顿回路，每个点的度严格等于2，如果我们忽略了这个条件，那么这就变成了一个最小生成树(MST)问题了！具体的说，设总点集为P，假设我们已经得到了一部分路径$Q=\{p_1, p_2, p_3, \ldots, p_k\}$，那么我们以点集$P-Q+\{p_1, p_k\}$，求一个最小生成树，这就是剩下路径花费的一个下限。



实线表示已有的路径
虚线表示估价得到的MST

下面我选取了 50..100 的几组数据，对比了应用 MST 估价、随机调整两种方法和最优解的差距：



由图中可以看出，通过 MST 估价的程序得到的解于随机调整相比已有大大提高，而且随机调整比较容易陷入局部最优的情况，尽管可以通过类似遗传算法的方法进行偶尔的变异，但是效果仍然不是很好，可见估价函数的优势。

3.2 小结

对于求较优解一类的问题，往往题目本身是 NPC，或者很难在竞赛的时间内想到多项式解法，这个时候就需要选手们能随机应变。如果一个问题状态比较明显，便可以考虑将问题转化成一个比较容易求解的问题，作为我要解决问题的答案上限或下限。这里估价函数可以设计比较灵活，在时间和解的质量方面寻求平衡。

参考文献

[1]《Sokoban: Reversed Solving》Frank Takes

[2]《人工智能 复杂问题求解的结构和策略》 George F Luger

[3]《Incremental Evaluation of Minimum Spanning Trees for the Traveling Salesman Problem》 Anand Panangadan

感谢

感谢各位老师对我的培育指导

感谢与我一起训练的同学

感谢与我交流切磋的网上的各位朋友