浅谈启发式思想在信息学竞赛中的应用

绍兴市第一中学 任之洲

摘 要

启发式算法是基于经验设计的算法，相较于传统的最优化算法，启发式往往通过牺牲精确性或稳定性以换取速度的提升。本文试图给出启发式算法在信息学竞赛中的进一步定义，并探讨一些常用的启发式算法和一些问题模型，其中包括一种最小斯坦纳树的$O(|S|\ |V|^2)$的近似解算法，并证明了该算法可以做到$2(1-\frac{1}{leaf})$近似。

1 引言

启发式算法是相对于最优化算法提出的一类算法，是基于直观或经验构造的算法，目的是在可接受的时间、空间开销下得到问题的一个较优解。这个概念非常宽泛，大多数启发式算法都十分复杂，并不能直接应用到OI竞赛中。

笔者对一些启发式算法进行简化和改进，使其能够适用于OI竞赛。并对于OI竞赛中的启发式算法，进一步给出了定义：

- 以直观感受为依据对算法进行调整优化，牺牲算法的稳定性换取速度提升的算法。
- 以直观感受为依据直接构造较优解，牺牲算法的正确性完成快速求解的算法。

在一些常用算法中也包含着启发式思想，这些算法经过优化后虽然没有降低最坏复杂度，但是算法效率都有了普遍的提升，比如：

- *k-d tree*最近点查询。
- 队列优化的*Bellman-Ford*算法（*SPFA*算法）及其贪心优化。
- 模拟退火算法、遗传算法等由自然科学衍生得到的随机搜索算法。

本文将对三个启发式算法进行探讨：

- A*算法及其变种，利用A*算法可以搭建起近似化算法和最优化求解的桥梁。
- 朴素贝叶斯分类，将直观判断和数学推导相结合的概率分类算法。
- 启发式合并，一种简单的合并类问题优化算法。

除了套用已有的经典模型，在很多时候还可以自己着手设计启发式算法，本文耗费了大量篇幅介绍了两个经典问题的启发式算法：

- 集合带权均分问题(2-*partition problem*)，得出了一个正确率很高的近似化算法，其设计流程可以给以启迪。
- 斯坦纳树问题(*steiner tree problem*)，耗费了较大篇幅介绍，得出了一个近似偏差程度可估计的高效算法，通过结合A*算法模型进一步得出了k小斯坦纳树近似解的高效算法。

对本文的每一部分，作者都给出了例题，希望能起到抛砖引玉的作用。

2 A*算法

在一些问题中，很难找到有效的多项式复杂度算法，只能采取搜索策略，但问题的状态数量往往非常的多。这时即可考虑设计估价函数进行剪枝，本文接下来将给出一些利用估价提高算法效率的方法。

2.1 启发式搜索

设当前状态为s，到状态s的实际代价为$g(s)$，从状态s到目标状态的估价为$h(s)$，实际最优代价为$h^*(s)$。

设$f(s) = g(s) + h(s)$，可以利用这个估价，每次选取$f(s)$最优[1]的状态扩展，并根据估价的性质进行剪枝。

2.2 $h(s) = h^*(s)$

当估价函数可以准确得出精确的代价时，每次选取$f(s)$最优的状态扩展，将可以快速得到最优解。

在得到最优解之后，继续扩展剩下的状态将可以进一步得到k优解，且避免了多余状态的计算。对于解法的一般化和特殊问题的进一步优化请参见俞鼎力的2014集训队论文。

[1]本文默认问题为最小化代价。

2.3 $h(s) \leqslant h^*(s)$

有时候为了方便计算，会舍去题目中的一些限制来进行估算，得到一个“优”于最优解的粗略解。

这时估价能够提供准确的剪枝依据，减少搜索状态的范围。

2.3.1 启发式迭代加深

启发式迭代加深算法（IDA*）是A*算法的一个变种，结合了迭代加深思想和估价函数，在一些状态量大、存储困难的问题中可以大量节省空间需求，书写代码也更加简洁。

在朴素的迭代加深算法中，每次会选择固定的步长进行搜索，在搜索过程中利用搜索深度限制进行剪枝。

在启发式迭代加深算法中，通过计算估价来进行剪枝。由于$h(s) \leq h^*(s)$，计算被剪掉状态最小的$g(s) + h(s)$可以估计下一个可能的答案取值，减少固定步长迭代的无用计算。

2.3.2 例题一

例1 (UVa 10181 15-Puzzle Problem)**.**

十五数码问题是八数码问题的一个扩展，在$4 * 4$的方格中有15个带编号格子和一个空格子，每次可以把空格子四周的一个格子移动到空格子位置。

A random puzzle position | The missing Tile moves to right. Denoted by R | The missing Tile moves upwards. Denoted by U | The missing Tile moves to the left. Denoted by L

要求在最小步数移动到目标状态，数据保证除了无解情况都能在45步内找到解。

对于无解的判断需要对数码问题进行进一步分析，本文不做深入讨论。

数码问题可以设计一种很简单的估价，求出所有非空格子到目标位置的曼哈顿距离之和作为$h(s)$，相当于忽略了只能和空格子交换的限制，也忽略了交换操作对另一个格子的影响。

由于忽略了一些限制，所以$h(s) \leq h^*(s)$，对于保证45步内出解的数据剪枝效果和迭代次数的优化效果都是不错的。

2.4 Anytime algorithm

也称可中断算法(interruptible algorithm)。

并不是所有问题都容易找到高效的估价方式，有时候只能对答案进行粗略估计，无法控制$h(s) \leq h^*(s)$，也就无法有效缩小搜索范围。在这种情况下，该问题则往往以提交答案题的方式出现，可以选用逐步变优的算法，在比赛时间内尽可能地得到较优的解。

2.4.1 例题二

例2 (CTSC 2009 Day2 N^2数码游戏)**.**

N^2数码游戏由N^2-1个滑块和一个空格组成，目标是通过上下左右移动空格使得N^2-1个滑块排成某一种特定的顺序。

提交答案题类型，$3 \leq N \leq 6$，共10组数据[2]

相较与例 1 ，这一题的数据范围大了许多，但是与之相对应的该题也不再要求输出精确解，只要能在比赛的许可时间内得到较优解就可以获得可观的分数。

直接套用例 1 中的估价用IDA*搜索，可以在短时间内得到其中5个测试点的最优解，但是这个估价计算出的值和精确的答案还是存在很大的差距的，对于剩下的测试点就无能为力了。

有一个简单的解决方法，设原估价函数为$h(s)$，得到新的估价$h_2(s) = ch(s)$，其中c为一个常数，根据具体测试点调整。

和$h(s)$相比，$h_2(s)$能更接近地估计真实步数，考虑将其套用在A*算法上。但是A*算法的内存开销非常大，这也是在例 1 中不选用A*的原因。

对于提交答案题，可以采用一种折中的手段，只保留估价最好的一部分状态，每次选择最佳状态扩展，如果新状态存储不下就舍弃最差的状态。由于该问题的解法方案很多，所以舍弃一些“差”的状态对最终答案的影响不大，除了一个特殊点都可以获得9 ~ 10分。

[2]具体分布：1组$N=3$，4组$N=4$，4组$N=5$，1组$N=6$。

3 启发式合并

启发式合并是一个简单的合并策略，可以有效优化合并算法的效率。

3.1 数据结构的启发式合并

假设有两个数据结构需要合并，这两个数据结构A、B维护的元素数量分别为a和b（$a \le b$），我们选择把数据结构A中维护的所有信息单独拆出，逐一插入到数据结构B 中。

定理3.1. 设经过若干次合并操作后得到了一个元素数量为n的数据结构，那么启发式合并策略所造成的插入操作次数是$O(n \log n)$的。

Proof. 对数据结构中维护的每个元素单独考虑，当一个元素被插入到另一个数据结构中时，它所在数据结构中维护的元素量至少变为原来的2倍，经过$O(\log n)$次插入操作后，就必定已在最终的数据结构中了。 □

启发式合并策略可以完成一些结构复杂的数据结构的合并，也可以简单拓展到其他数据集的合并中。

3.1.1 例题三

例3 (HNOI2009 梦幻布丁)**.**

n个布丁摆成一行，进行m次操作。

每次将某个颜色的布丁全部变成另一种颜色的，然后再询问当前一共有多少段颜色。

数据范围：$n \le 10^5$

可以用链表或*vector*等数据结构维护每种颜色的布丁，修改时暴力拆解较小的链表或*vector*，逐个修改计算贡献。

根据定理 3.1 ，修改次数有$O(n \log n)$保证，暴力维护即可。

3.1.2 例题四

例4 (BZOJ3510 首都)**.**

维护一个n个点的森林，共三种操作：

- 连接点u和点v，保证连接后仍为森林。
- 询问点u所在树的重心（若有两个点可作为重心，选择编号较小的）。
- 询问所有树重心编号的异或和。

先考虑给一棵树增加一个叶子节点，重心只会往新叶子节点的方向移动，且最多移动一次。

当连接两棵树时，可以把点数较小的树暴力拆分，以叶子节点的形式加入，同时维护重心。根据定理 3.1，加入叶子的次数有$O(n\log n)$保证，只需另维护一个数据结构用于判断重心是否移动。

3.2 并查集的按秩合并

按秩合并是一个实用的并查集维护方式，每次选择把较小集合的根接到较大集合的根下，根据定理 3.1 易得树高为$O(\log n)$。具体实现时，也可以直接维护树深度按秩合并。

按秩合并可以在不进行路径压缩的情况下保证并查集的树深度，在一些问题中可以避免路径压缩时的信息缺失。

3.2.1 例题五

例5 (NOIP2013 货车运输)**.**

有n座城市，编号$1\sim n$，城市之间有m条双向道路，每一条道路对车辆都有重量限制z_i，简称限重。

现在有q辆货车在运输货物，司机们想知道每辆车在不超过车辆限重的情况下，最多能运多重的货物。

数据范围：$n\le 10000$，$m\le 50000$，$q\le 30000$，$z_i\le 100000$

考虑用$kruskal$求出最大生成树，最大生成树上的路径最小值就是这两个点连通的最宽限重，可以用倍增求解，时间复杂度$O(m\log m+q\log n)$。

如果用按秩合并代替路径压缩，可以记录并查集中每个点和它父亲连通时的边的限重，路径最小值仍可以表示两个点连通的最宽限重。暴力求解的时间复杂度为$O(m\log m+q\log n)$，比前一做法简洁很多。

由于转化后的问题仍是路径最小值，所以仍然可以采用倍增，询问部分理论上可以做到$O(q \log\log n)$。若选用线性的排序方法，并对并查集进行路径压缩（仍按原方法连边），时间复杂度理论上可以做到$O(z + m\alpha(n) + q \log\log n)$。

4 抽象问题与机器学习

当我们遇到一些抽象的分类问题时，直观的感觉并不能提供准确的判断依据，这时可以考虑直接采用机器学习代替人工观察。

本文接下来将通过一道GCJ题，简单介绍一种实用的分类算法。

4.1 朴素贝叶斯

4.1.1 贝叶斯公式

设$P(A \mid B)$表示事件B已经确定发生的前提下，事件A发生的概率，$P(A \cap B)$表示事件AB都发生的概率。

定理4.1 (贝叶斯公式)**.**

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Proof.

$$P(A \cap B) = P(A)P(B \mid A) = P(B)P(A \mid B)$$

移项后即可得贝叶斯公式。 □

4.1.2 朴素的判断

设S为待分类事件，A和B为可选择的分类。

若$P(A \mid S) > P(B \mid S)$，则认为$S$属于$A$类，否则属于$B$类。

当有更多的分类时，则选择概率最高的一项。

4.1.3 朴素的假定

设待分类事件$S = \{S_1, S_2, ..., S_m\}$，$S_i$为特征属性。

假定这些特征属性是相互独立的，可以得到

$$P(A \mid S) \approx \prod_{i=1}^{m} P(A \mid S_i)$$

通过比较这样近似计算的概率来进行分类。

$P(A \mid S_i)$的计算可以通过随机抽样或设计其他算法来完成。

4.2 例题六

例6 (GCJ Round 1A 2014 Proper Shuffle)**.**

有两种生成排列的方式，并给出120个长度为1000的排列，每个排列都是由这两个算法之一生成的，且这些排列都是独立生成的。

现在要求识别这些排列是由哪个算法生成的，对120个排列中的109个及以上输出正确解则判定为AC。

两种算法的流程如下：

Algorithm 1 GOOD

```
for k = 0 to n − 1 do
    a_k = k
end for
for k = 0 to n − 1 do
    p = random_int(k..n − 1)
    swap(a_k, a_p)
end for
```

Algorithm 2 BAD

```
for k = 0 to n − 1 do
    a_k = k
end for
for k = 0 to n − 1 do
    p = random_int(0..n − 1)
    swap(a_k, a_p)
end for
```

其中$GOOD$算法可以等概率地生成排列，而BAD算法则不行。

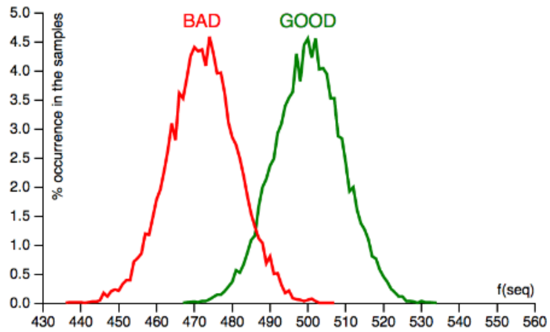
4.2.1 简单分析

对于$GOOD$算法，完成第k位的交换后，第k位上出现任何数的概率均为$\frac{1}{n}$，且之后的交换都不会影响这一位。

对与BAD算法，当决策第k位的交换时，$swap(a_k, a_p)$时$p < k$，这时原先在a_p的值就被交换到了更高的位上，并且此后会被交换到跟高的位上，这会对它出现在每位上的概率产生微妙的影响。

4.2.2 一种简单的特征判断

设S是输入的排列，$f(S)$为排列S中$S_i \leq i$的位置数，GCJ的Contest Analysis中给出了以下统计数据：



通过设定阈值判断可以达到90%以上的正确率。

4.2.3 朴素贝叶斯分类

设S是输入的排列，我们想要求出$P(GOOD|S)$，那么根据定理 4.1 易得

$$\begin{aligned} P(GOOD \mid S) &= \frac{P(S \mid GOOD)P(GOOD)}{P(S)} \\ &= \frac{P(S \mid GOOD)P(GOOD)}{P(S \mid GOOD)P(GOOD) + P(S \mid BAD)P(BAD)} \end{aligned}$$

因为$P(GOOD) = P(BAD)$，所以式子可以简化

$$P(GOOD \mid S) = \frac{P(S \mid GOOD)}{P(S \mid GOOD) + P(S \mid BAD)}$$

同时也可以得到

$$P(BAD \mid S) = \frac{P(S \mid BAD)}{P(S \mid GOOD) + P(S \mid BAD)}$$

假如$P(GOOD \mid S) > P(BAD \mid S)$，那么$S$由$GOOD$算法生成的概率较大。容易发现$P(GOOD \mid S) > P(BAD \mid S)$的条件是$P(S \mid GOOD) > P(S \mid BAD)$。

由于$GOOD$算法能等概率生成所有排列，所以

$$P(S \mid GOOD) = \frac{1}{n!}$$

但是$P(S \mid BAD)$的计算则很难完成，于是选择近似化处理：

$$P(S \mid BAD) \approx \prod_{k=0}^{n-1} P(S_k \mid BAD)$$

为了使比较概率时更加公平，我们对能精确计算的$GOOD$算法也进行相同的近似化处理：

$$P(S \mid GOOD) \approx \frac{1}{n^n}$$

设$P_k[i][j]$为BAD算法完成了k次交换后$S_i = j$的概率，那么容易设计一个$O(n^3)$的DP计算$P_n[i][j]$的值，也就得到了所需的$P(S_k \mid BAD)$的值。

计算出$P(S \mid BAD)$后与$P(S \mid GOOD)$比较即可[3]。

5 带权均分问题

例7. 设S为一个由n个元素构成的多元集合(*multiset*)，$f(S)$为S 集合中所有元素之和。将集合S划分为两个集合S_1、S_2，使$S_1 + S_2 = S$，求$|f(S_1) - f(S_2)|$的最小值，并求出划分方案。

5.1 简单举例

设$S = \{3, 1, 1, 2, 2, 1\}$，下面提供两种合法的划分方案使得$f(S_1) = f(S_2) = 5$，$|f(S_1) - f(S_2)| = 0$：

- $S_1 = \{1, 1, 1, 2\}$ $S_2 = \{2, 3\}$
- $S_1 = \{3, 1, 1\}$ $S_2 = \{2, 2, 1\}$

[3]直接计算$P(S \mid BAD)$涉及高精度运算，可以考虑计算$P(S \mid BAD)n^n$与1.0比较。由于经过了近似化处理，所以也可以用其他阈值代替1.0。

5.2 一种动态规划算法

可以设计一个动态规划，类似背包问题，设$f[i][j]$ 表示使用了前i个元素，$f(S_1) = j$的方案是否可行，时间复杂度为$O(nf(S))$。

当集合S中的元素都很小时，这确实是一个很优秀的算法，但这其实是一个伪多项式复杂度算法(pseudo-polynomial time algorithm)。

5.3 一种简单的贪心思想

当我们拿到这个问题时，很容易产生这样的想法：把元素读入后，依次把每个元素放入当前和较小的集合中。

为了使两个集合的大小逐渐逼近，所以把元素排成降序处理，时间复杂度$O(n\log n)$。

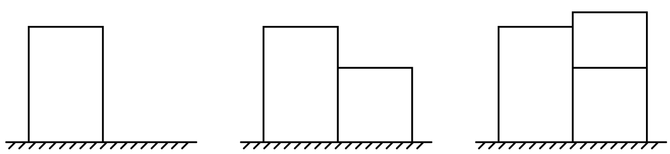
这个算法的思路十分自然，而且效果也不错，但是经过仔细分析后会发现是存在明显漏洞的。

5.4 差分算法

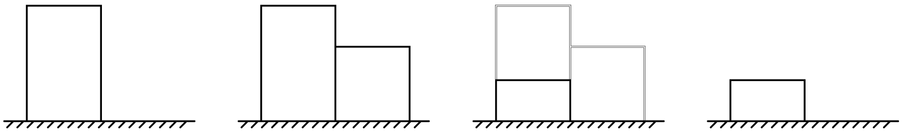
差分算法是对前一节中提到的贪心算法的改进，思路也很简洁，近似解效果有了很大的优化。

5.4.1 基本思想

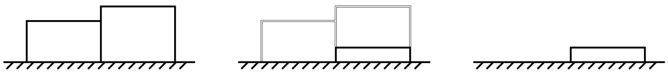
仔细考虑前一算法，该贪心算法的流程如下图：



考虑对算法进行修改，当我们把两个数X、Y放到不同集合时，对答案产生影响的部分只有两数的差的大小，相当于得到了一个新数$|X-Y|$，如图:



对于新得到的这个数，仍可以把它当作一个普通的数看待，继续进行原算法。



重新回顾前一节的贪心算法，考虑每一次决策时的状态。设新放入的元素为X，已完成决策的元素得到的差分值为Y。把新元素放入较小集合这个操作，相当于把X 和Y替换为$|X-Y|$，而我们设计这个算法的初衷是优先处理较大的元素，所以依次决策是不优的。

5.4.2　算法流程

根据差分思想，可以得出一个新的算法，需要用到堆和并查集[4]维护。

- 把集合S中的所有元素放入堆中，并查集中每个元素是一个独立集合。
- 从堆中取出最大的两个元素，设为X、Y，对应并查集中为集合S_X、S_Y。这时做出决策，把$|X-Y|$ 放入堆中，并查集中合并集合S_X、S_Y，连边时连接权值为1 的边。
- 当堆中只剩下一个元素时，进入下一步，否则重复前一步。
- 设并查集树结构的根在S_1集合，其它元素根据到根路径长度的奇偶性判断所在集合（偶→ S_1，奇→ S_2）。

[4]这里的并查集实现可以在路径压缩的时候维护路径长度的奇偶性，或者直接选用按秩合并保证$O(\log n)$树高。

5.4.3 近似效果

这个算法的时间复杂度为$O(n\log n)$，是非常高效的，所以主要需要关注它得出的解是否够优。

当n不过于小时，比如假设$n \geq 1000$，对于随机生成的数集$S\,(S_i \in [0, 2^{31}))$，该算法能以接近100% 的概率划分出集合S_1、S_2使得$|f(S_1) - f(S_2)| \leq 1$，也就是$f(S)$是奇数时$|f(S_1) - f(S_2)| = 1$，$f(S)$是偶数时$|f(S_1) - f(S_2)| = 0$。可以看出这个算法的正确率是相当高的。

在n较小的时候可以利用该算法设计估价函数，进行启发式搜索来获得更优的解。

5.5 例题八

例8 (POI2013 Polarization)**.**

给出一个n个点的树，把树上的每条边定向为单向边。若点u 可以到达点v，则称(u, v)为合法点对。求出所有定向方案中合法点对数量的最小值和最大值。

数据范围：$n \leq 250000$

5.5.1 问题转化

树可以表示为二分图，所以最小值一定为$n - 1$。

对于最大值，最优解一定是存在一个点u使得剩下的点要么存在它到u的路径，要么存在u到它的路径。进一步分析易得u一定选取在树的重心位置。

以u为根划分出若干子树，剩下的问题就是把这些子树划分成两个集合使其大小的乘积最大。

5.5.2 传统最优化算法

注意到这里$f(S) = n$，所以说不同的元素种数只有$O(\sqrt{n})$ 种，转化为多重背包问题。

多重背包是经典问题，这里不详细介绍。

时间复杂度$O(n\sqrt{n})$。

5.5.3 近似化算法

假如直接使用差分算法计算解，能够通过80%的数据，主要原因在于当元素个数较少时，近似解的偏差率较大。

可以定一个阈值，当子树个数小于这个阈值时，采用暴力背包算法。

经过这样的拼接后将可以通过全部POI官方数据，时间复杂度$O(n \log n)$。

在传统最优化算法运行效率良好时并不提倡使用近似化算法，这里只是为了说明差分算法能够得出很优的近似解。

6 最小斯坦纳树问题

例9. 给出一张带权无向图$G = (V, E)$以及一个集合S，V表示图G的点集，E表示图G的边集，S是V的一个子集。求一个连通图$T = (V_T, E_T)$，满足$S \subseteq V_T \subseteq V$，$E_T \subseteq E$，$|E_T| = |V_T| - 1$，并使得边集$E_T$的权值和最小。

6.1 一种最优化算法

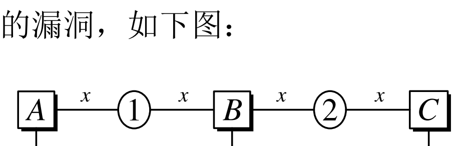
最小斯坦纳树有经典的状态压缩算法，转化为最短路问题模型，具体请参见姜碧野的2014集训队论文。

时间复杂度$O(|V|\ 3^{|S|} + |E|\ 2^{|S|})$，这是一个很优秀的算法，但是能适用的数据范围很小。

6.2 一种简单的贪心思想

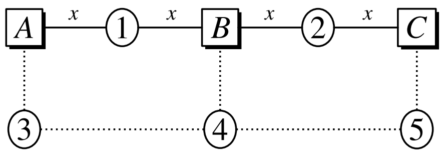
有一个简单的贪心：对原图求一棵最小生成树，再剔除最小生成树上多余的边，即在不影响点集S连通性的前提下尽可能地删边。

这个算法存在很大的漏洞，如下图：

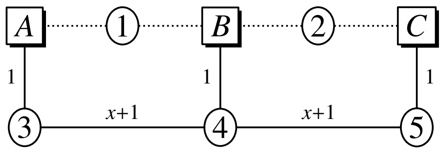


设$V = \{A, B, C, 1, 2, 3, 4, 5\}$，$S = \{A, B, C\}$，$x \geq 2$，直接采用上述贪心得到的

解如下图：



容易发现最优解应为下图：

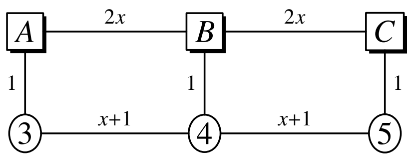


6.3　启发式算法

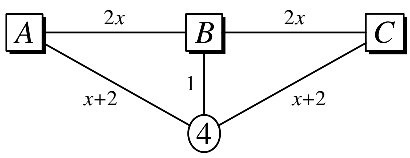
根据上面那个例子，可以对算法进行一些针对性的改进。

6.3.1　进一步分析

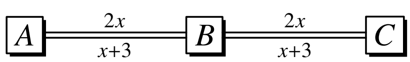
在这个例子中，假设忽视点1和点2可以得到下图：



对这张图做最小生成树已经可以得到最优解，但是为了使问题更一般化，考虑把剩下的非S集节点也从图上去除。

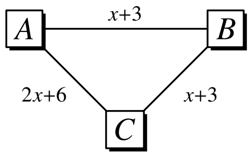


假如把点4也从图上去除，就得到下图：



容易发现一个问题，两条权为$x + 3$的边有交，所以在最后做出的生成树中要除去重复统计的边。

为了将算法更一般化，可以构出一张完全图G'，图G'中只保留集合S中的点，边权为原图中的最短路长度。



6.3.2 算法流程

经过进一步整理后可以得到如下算法：

- 由原图$G = (V, E)$构造完全图$G_1 = (S, E_1)$，E_1边权为G中最短路长度。
- 得到G_1的最小生成树T_1（如果有多解则随意选择一个）。
- 构造G的一个子图$G_2 = (V_2, E_2)$，其中$V_2 = V$，E_2为T_1中的边在G中的路径（如果有多条最短路则随意选择一条）。
- 得到G_2的最小生成树T_2（如果有多解则随意选择一个）。
- 在T_2中去掉无用的边，使得所有叶子节点都是集合S中的点，得到斯坦纳树T。

分析算法每一步的时间复杂度：

- 构造完全图G_1需要完成$|S|$次单源最短路，复杂度$O(|S||V|^2)$[5]。
- 求最小生成树T_1，完全图宜选用$Prim$算法，复杂度$O(|S|^2)$。
- 构造图G_2，每条边所对应的最短路可能会经过$O(|V|)$个点，但是所有路径边数相加是$O(|V|)$的。
- 求最小生成树T_2，复杂度$O(|V|^2)$或$O(|V|\log|V|)$。
- 删去无用边，复杂度$O(|V|)$。

可以看出这个算法的运行效率是十分高效的，而其近似化效果也十分优秀，下文将给出对其近似解的相对上界证明。

[5]可以选用高效的最短路算法进一步优化复杂度。

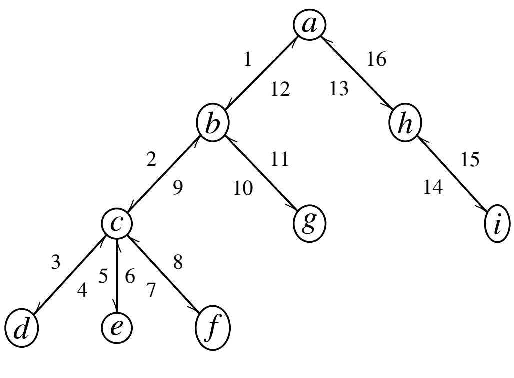
6.3.3 近似效果

设D_T为上述算法得到的斯坦纳树T的边权和，D_{MIN}为最小斯坦纳树T_{MIN}的边权和，下文将分析得出$\frac{D_T}{D_{MIN}}$的上界。

引理6.1. 对于一棵有$m(m \geq 1)$条边的树T，可以在树上找到一个环，$u_0, u_1, u_2, \ldots, u_{2m}$，其中$u_0 = u_{2m}$，每个点$u_i(0 \leq i \leq 2m)$ 都是T中的一个节点，点对$(u_{i-1}, u_i)(1 \leq i \leq 2m)$在$T$中有边，且这个环满足以下性质：

- T中的每条边在环中正好出现两次。
- T中的每个叶子节点在环中只出现一次（不考虑u_0）。
- 假设u_i和$u_j(0 < i < j)$是T中的两个叶子节点，且不存在叶子节点$u_k(i < k < j)$，那么$u_i, u_{i+1}, \ldots, u_j$ 是T 上的一条简单路径。

Proof. 选定一个点作为根节点后，对树T进行欧拉遍历后得到的欧拉遍历序[6]满足上述性质。



□

定理6.1. 设$leaf$为最小斯坦纳树T_{MIN}中叶节点的个数，则

$$\frac{D_T}{D_{MIN}} \leq 2\left(1 - \frac{1}{leaf}\right) \leq 2\left(1 - \frac{1}{|S|}\right)$$

[6]把无向边视作两条方向相反的有向边后，以根为起点的欧拉回路。

Proof.

因为$leaf \le |S|$，所以显然$2\left(1-\frac{1}{leaf}\right) \le 2\left(1-\frac{1}{|S|}\right)$。

根据引理 6.1 ，可以在T_{MIN}中找出一个环L，设环L的边权和为D_L，则$D_L = 2D_{MIN}$。

在环L上选择最长的一条叶子节点间的简单路径删去后，得到一条路径P，设路径P的边权和为D_P，则易得$D_P \le \left(1-\frac{1}{leaf}\right)D_L$，即$D_P \le 2\left(1-\frac{1}{leaf}\right)D_{MIN}$。

由于T中两点间都选取了最短路径，所以$D_T \le D_P$，即$D_T \le 2\left(1-\frac{1}{leaf}\right)D_{MIN}$。

□

定理6.2. 设$leaf$为最小斯坦纳树T_{MIN}中叶节点的个数，在$leaf \ge 2$时，最坏情况下

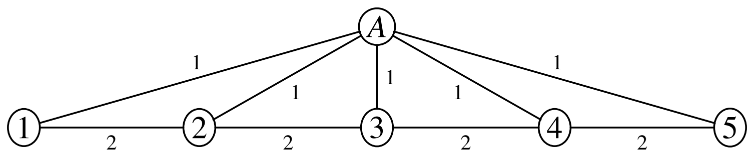
$$\frac{D_T}{D_{MIN}} = 2\left(1-\frac{1}{leaf}\right)$$

Proof.

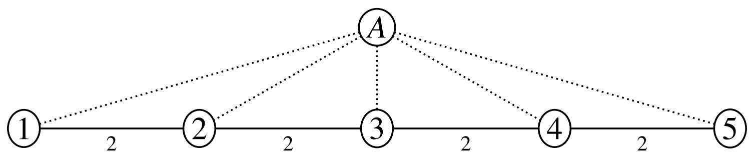
可以这样构造图$G = (V, E)$，$V = \{v_1, v_2, \dots, v_{leaf+1}\}$，$S = \{v_1, v_2, \dots, v_{leaf}\}$，边集定义如下：

$$d(v_i, v_j) = \begin{cases} 2 & i+1 = j \ \ and \ \ i < leaf \\ 1 & i \le leaf \ \ and \ \ j = leaf+1 \\ \infty & otherwise \end{cases}$$

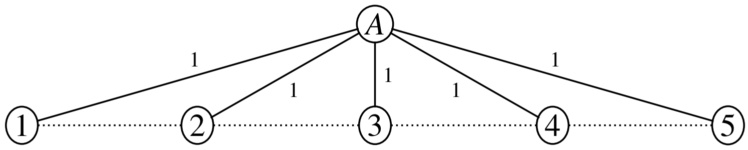
如下图，$V = \{A, 1, 2, 3, 4, 5\}$，$S = \{1, 2, 3, 4, 5\}$：



在最坏情况下，T的构成会是这样：



然而，T_{MIN}的构成是这样：



对于这样构造的图，比值达到了上限

$$\frac{D_T}{D_{MIN}} = \frac{2(leaf-1)}{leaf} = 2\left(1 - \frac{1}{leaf}\right)$$

□

6.4 例题十

例10 (2015集训队互测 Marketing network)**.**

给出图$G = (V, E)$，求前k小生成森林，要求特定点集S在生成森林中连通。

数据范围：$|V|, k \le 50$，$|E| \le 100$，$|S| \le 15$，图的边、边权以及点集S均随机生成。

6.4.1 最优化算法

对于第k优解问题，可以用A*算法求解。

可以用每条边是否选取来描述状态，搜索时确定下一条边是否选取。

选用斯坦纳树的状态压缩算法可以做到$h(s) = h^*(s)$，每次选择最优状态扩展，由于要求的是生成森林，所以每$O(n)$次有效决策就能找到一个当前最优解，求前k优解只需要进行$O(kn)$次估价计算。

这个算法能够保证正确性和复杂度，但效率较低，并不能胜任该题数据范围。

6.4.2 近似化算法

继续沿用A*思想，选用前文的启发式算法作为估价，但不幸的是$h(s) \ge h^*(s)$，虽然能估计出下界，但是不能有效剪枝。

考虑进行近似化处理，选定一个常数c，把$ch(s)$作为$h^*(s)$的下界参考，剪去估价不优的状态，通过调整常数c可以平衡运行效率和正确性。

7 感谢

感谢计算机协会提供学习和交流的平台。

感谢绍兴一中的陈合力老师、董烨华老师、邵红祥老师、游光辉老师多年来给予的关心和指导。

感谢国家集训队教练余林韵和陈许旻的指导。

感谢清华大学的俞鼎力、董宏华、何奇正学长对我的帮助。

感谢绍兴一中的张恒捷、王鉴浩、贾越凯同学对我的帮助和启发。

感谢其他对我有过帮助和启发的老师和同学。

参考文献

[1] Wikipedia, Heuristic (computer science).

[2] Wikipedia, Partition problem.

[3] Wikipedia, Steiner tree problem.

[4] L Kou, G Markowsky, L Berman, “A fast algorithm for Steiner trees”, Acta Informatica.

[5] 姜碧野, 迭代求解的利器——SPFA 算法的优化与应用, 2009集训队论文。

[6] 周而进, 浅谈估价函数在信息学竞赛中的应用, 2009集训队论文。

[7] 俞鼎力, 寻找第k优解的几种方法, 2014集训队论文。