

爬山法（深度优先的扩展）

重点：按启发测度从大到小的顺序压入

缺点：有时会陷入局部极值

• 爬山法算法

1. 构造由根组成的单元素栈 S ;
2. If $Top(S)$ 是目标节点 Then 停止;
3. Pop(S);
4. S 的子节点按照其启发测度由大到小的顺序压入 S ;
5. If S 空 Then 失败 Else goto 2.

Best-First（广度优先的扩展）

重点：评价函数，使用堆(其实使用最小优先队列)

• Best-First 搜索算法

1. 使用评价函数构造一个堆 H ,首先构造由根组成的单元素堆;
2. If H 的根 r 是目标节点 Then 停止;
3. 从 H 中删除 r ,把 r 的子节点插入 H ;
4. If H 空 Then 失败 Else goto 2.

分支界限

重点：利用一定条件进行剪枝，减少搜索空间

• 计算解的代价的下界

命题2. 把代价矩阵某行(列)的各元素减去同一个数,不影响优化解的求解.

- 代价矩阵的每行(列)减去同一个数(该行或列的最小数),使得每行和每列至少有一个零,其余各元素非负.

解空间的下界就是减去的公共部分的总和

• 分支界限搜索(使用爬山法)算法

1. 建立根节点,其权值为解代价下界;
2. 使用爬山法,类似于拓扑排序序列树生成算法求解问题,每产生一个节点,其权值为加工后的代价矩阵对应元素加其父节点权值;
3. 一旦发现一个可能解,将其代价作为界限,循环地进行分支界限搜索:剪掉不能导致优化解的解,使用爬山法继续扩展新增节点,直至发现优化解.

TSP问题

转换为树搜索问题

- 所有解集合作为树根，其权值由代价矩阵使用上节方法计算；
- 用爬山法递归地划分解空间，得到二叉树
- 划分过程：
 - 选择图上边 (i, j) 使右子树代价下界增加最大
 - 所有包含 (i, j) 的解集合作为左子树
 - 所有不包含 (i, j) 的解集合作为右子树
 - 计算出左右子树的代价下界

A*

A*算法计算过程中无需剪枝？

A*算法的基本思想

- A*算法与分支界限策略的比较
 - 分支界限策略是为了剪掉不能达到优化解的分支
 - 分支界限策略的关键是“界限”
 - A*算法的核心是告诉我们在某些情况下，我们得到的解一定是优化解，于是算法可以停止
 - A*算法试图尽早地发现优化解
 - A*算法经常使用Best-first策略求解优化问题

• A*算法关键—代价函数

- 对于任意节点 n
 - $g(n)$ = 从树根到 n 的代价
 - $h^*(n)$ = 从 n 到目标节点的优化路径的代价
 - $f^*(n) = g(n) + h^*(n)$ 是节点 n 的代价
- What is the value of $h^*(n)$?
 - 不知道！
 - 于是， $f^*(n)$ 也不知道
- 估计 $h^*(n)$
 - 使用任何方法去估计 $h^*(n)$ ，用 $h(n)$ 表示 $h^*(n)$ 的估计
 - $h(n) \leq h^*(n)$ 总为真
 - $f(n) = g(n) + h(n) \leq g(n) + h^*(n) = f^*(n)$ 定义为 n 的估计

- A*算法本质—已经发现的解是优化解

定理1. 使用Best-first策略搜索树, 如果A*选择的节点是标节点, 则该节点表示的解是优化解.

证明. 令 n 是任意扩展到的节点, t 是选中目标节点.

往证 $f(t)=g(t)$ 是优化解代价.

(1). A*算法使用Best-first策略, $f(t) \leq f(n)$.

(2). A*算法使用 $h(n) \leq h^*(n)$ 估计规则, $f(t) \leq f(n) \leq f^*(n)$.

(3). $\{f^*(n)\}$ 中必有一个为优化解的代价, 令其为 $f^*(s)$. 我们有 $f(t) \leq f^*(s)$.

(4). t 是目标节点 $h(t)=0$, 所以 $f(t)=g(t)+h(t)=g(t) \leq f^*(s)$.

(5). $f(t)=g(t)$ 是一个可能解, $g(t) \geq f^*(s)$, $f(t)=g(t)=f^*(s)$.

A*算法的规则

- (1). 使用Best-first策略搜索树;
- (2). 节点 n 的代价函数为 $f(n)=g(n)+h(n)$,
 $g(n)$ 是从根到 n 的路径代价, $h(n)$ 是从 n 到某个目标节点的优化路径代价;
- (3). 对于所有 n , $h(n) \leq h^*(n)$;
- (4). 当选择到的节点是目标节点时, 算法停止, 返回一个优化解.