【cartographer】（2）分枝定界算法

EchoChou428 于 2021-03-24 10:55:51 发布

分枝定界方法

分枝定界法（branch and bound）是一种**求解整数规划问题**的最常用算法，是一种搜索与迭代的方法。

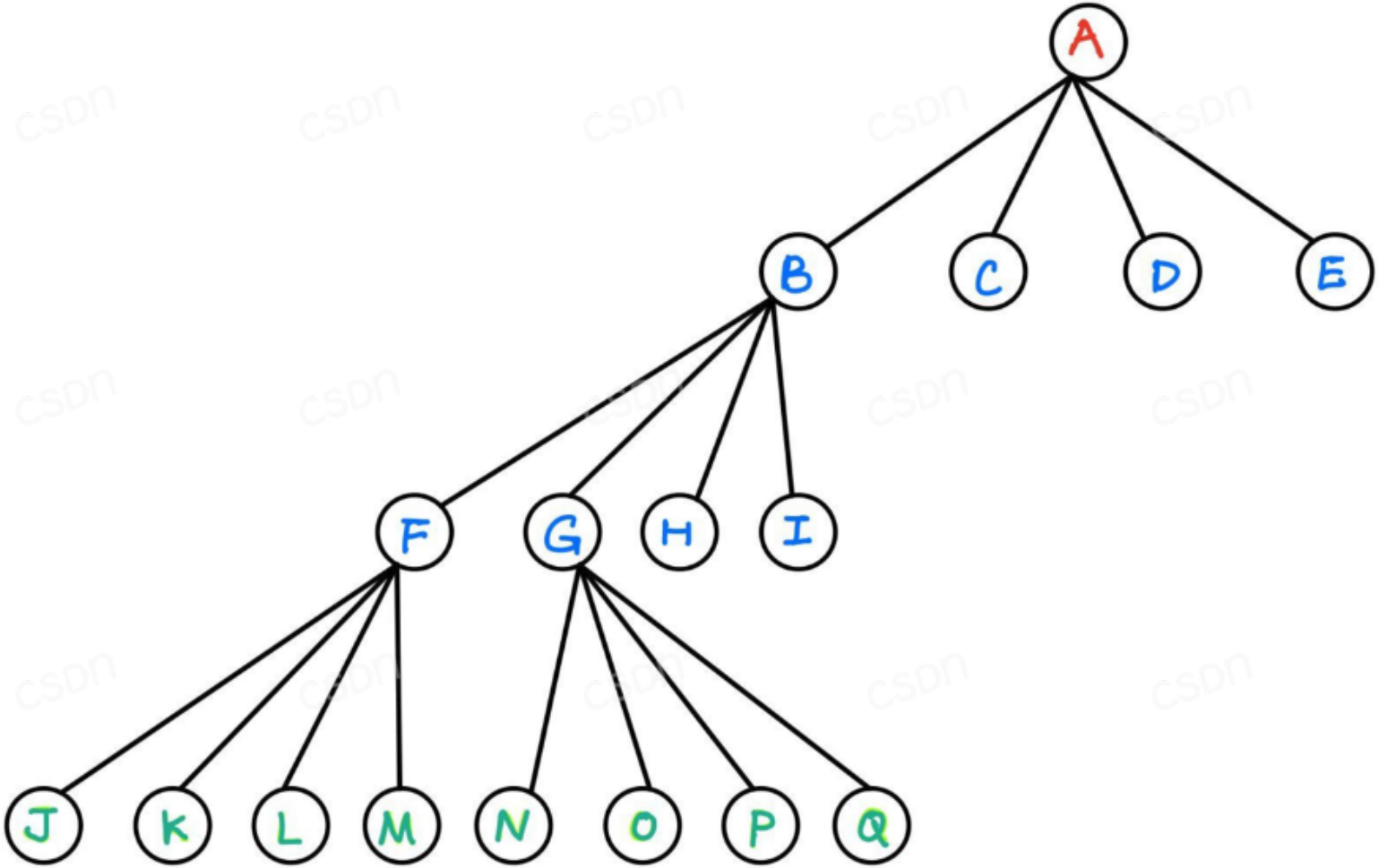
通俗说法：

分枝定界算法始终围绕着一颗**搜索树**进行的，主要流程就是分枝+定界。

我们可以将原问题看作搜索树的根节点，从这里出发，分枝定界的含义就是将大的问题分割成小的问题。大问题可以看成是搜索树的父节点，分割出来的小问题就是父节点的子节点，一直分割到最小问题为 叶子节点。**分枝**的过程就是不断给树增加子节点的过程。

而**定界**就是在分枝的过程中检查子问题的上下界（如果问题是找最大值就定上届，最小值就定下界），如果子问题不能产生比当前最优解还要优的解，那么砍掉整个的这一支。直到所有子问题都不能产生一个更优的解时，算法结束。

1. A为根节点，表示整个解空间ω
2. JKLMNOPQ为叶子节点，表示某一个具体的解
3. BCDEFGHI为中间节点，表示解空间ω的某一部分子空间



具体流程：

1. A分枝BCDE，按照优先级排序；
2. B分枝FGHI，按照优先级排序；
3. F分枝JKLM，按照优先级排序；
4. 在叶子节点上，计算JKLM的目标函数最大值，记为best_score（初始值为负无穷）；
5. 返回上一层G，计算G的目标函数值，与best_score比较：

若best_score依旧最大，则对G进行剪枝，继续计算H的目标函数值；

若G的目标函数值大于best_score，则对G进行分枝NOPQ，计算NOPQ的目标函数最大值，与best_score比较，更新best_score；

分枝定界在相关方法中的应用——结合多分辨率栅格

算法流程：1. 节点选择; 2. 分枝规则; 3. 上界计算

节点选择：

深度优先搜索：Depth-First-Search (DFS)

深度优先搜索属于图算法的一种，是一个针对图和树的遍历算法。深度优先搜索是图论中的经典算法，利用深度优先搜索算法可以产生目标图的相应拓扑排序表，利用拓扑排序表可以方便的解决很多相关的图论问题，如最大路径问题等等。一般用堆数据结构来辅助实现DFS算法。其过程简要来说是**对每一个可能的分支路径深入到不能再深入为止，而且每个节点只能访问一次**。

广度优先搜索（宽度优先搜索）：Breadth-First-Search(BFS)

广度优先搜索是连通图的一种遍历算法这一算法也是很多重要的图的算法的原型。Dijkstra单源最短路径算法和Prim最小生成树算法都采用了和宽度优先搜索类似的思想。属于盲目搜寻法，目的是系统地展开并检查图中的所有节点，以找寻结果。换句话说，它并**不考虑结果的可能位置，彻底地搜索整张图，直到找到结果为止**。基本过程，BFS是从根节点开始，沿着树(图)的宽度遍历树(图)的节点。如果所有节点均被访问，则算法中止。一般用队列数据结构来辅助实现BFS算法。

分枝规则：

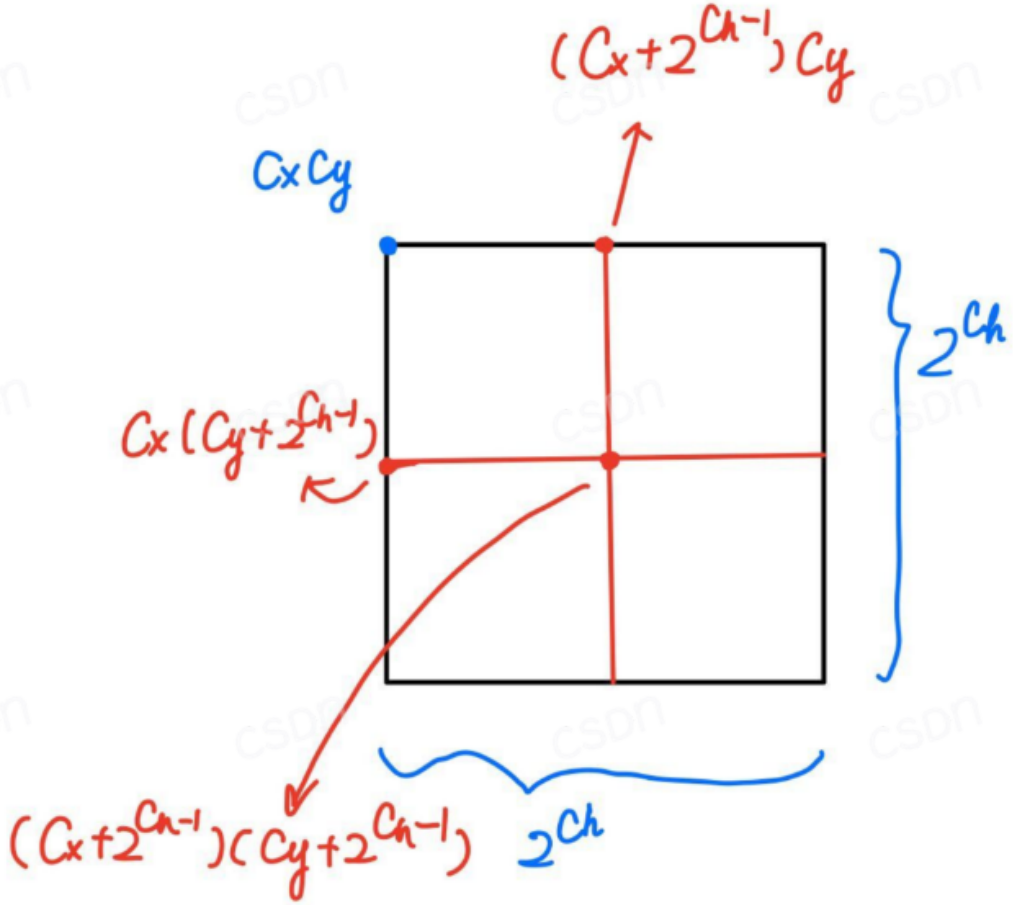
按照地图分辨率构造树形结构：假设有一张4×4的地图，我们降低其分辨率，每2×2的格子合并成一个，得到一张2×2的地图，再降低其分辨率，得到一张1×1的地图。这样，地图被分成了三层，如下图所示。

具体实现

分枝：先从根节点开始遍历，把它拆分成4个子格子。这4个子格子分别计算score，并由高到低排序。从中选出分数最高的格子，进一步拆分成4个子格子。假设这4个子格子已经到了叶子节点，也就是达到了真实地图的分辨率。此时计算出的4个子格子的score代表了真实的位姿评分，找出其最大值，记作best_score。**只有叶子节点的score是真实评分，其它格子的score都是上界。**

定界：第二层未曾探索的3个格子的score是否大于best_score，如果是，继续分支，当遇到score大于best_score的叶子节点时，更新best_score；如果否，直接将这个格子及其所有子格子剪枝。因为格子的评分代表了其子格子评分的上界，如果上界都小于best_score，就不可能再有子格子的评分大于它了。

搜索树中的每一个节点，都表示为一个正方形的搜索框。对于中间节点，分枝为四个子节点。



$C = (C_x, C_y, C_\theta, C_h)$

$C_c = (((\{C_x, C_x + 2^{C_h - 1}\} \times \{C_y, C_y + 2^{C_h - 1}\} \times C_\theta) \cap \overline{W}) \times \{C_h - 1\}$

$$\overline{\overline{W}}_c = \left(\left\{ (j_x, j_y) \in \mathbb{Z}^2 : \begin{array}{l} c_x \le j_x < c_x + 2^{c_h} \\ c_y \le j_y < c_y + 2^{c_h} \end{array} \right\} \times \{c_\theta\} \right),$$

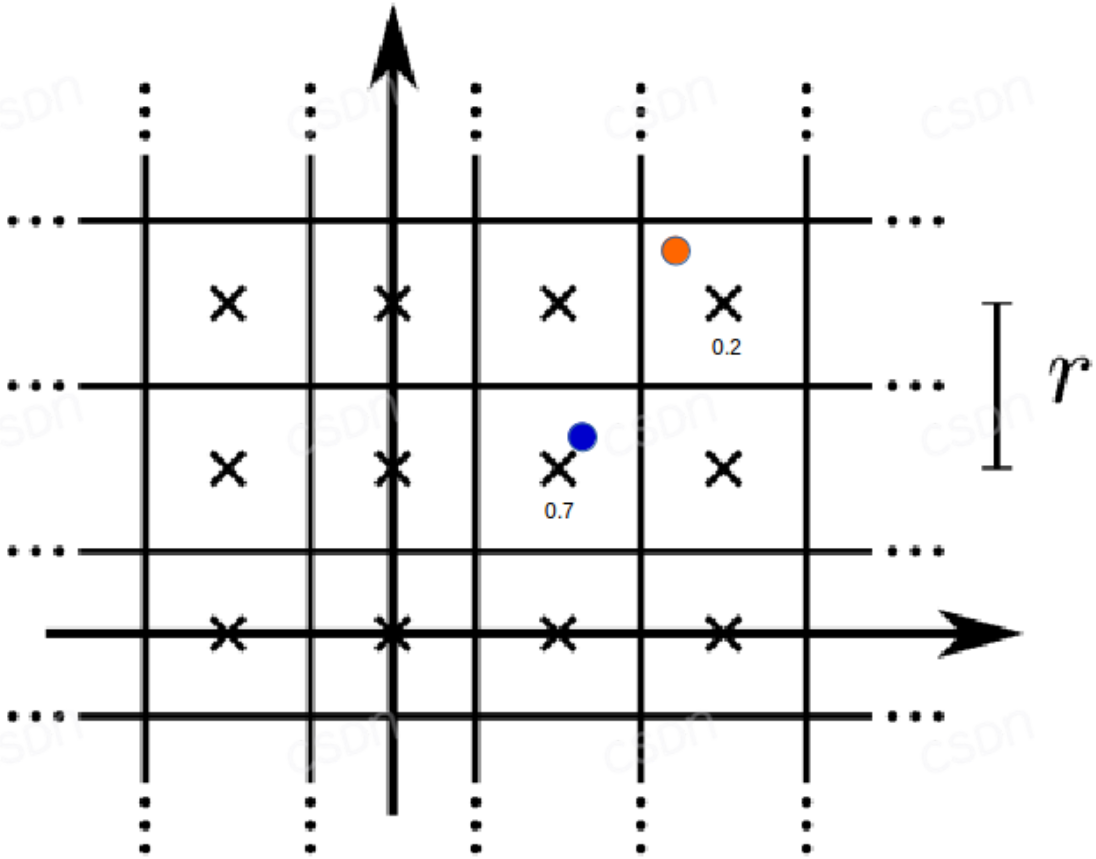
$$\overline{W}_c = \overline{\overline{W}}_c \cap \overline{W}.$$

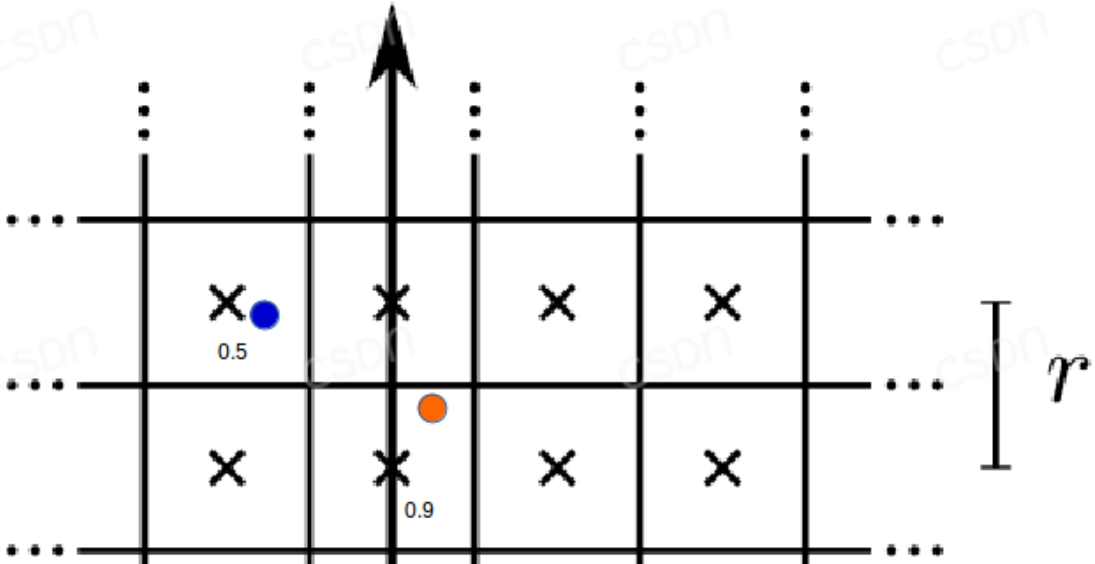
上界计算：

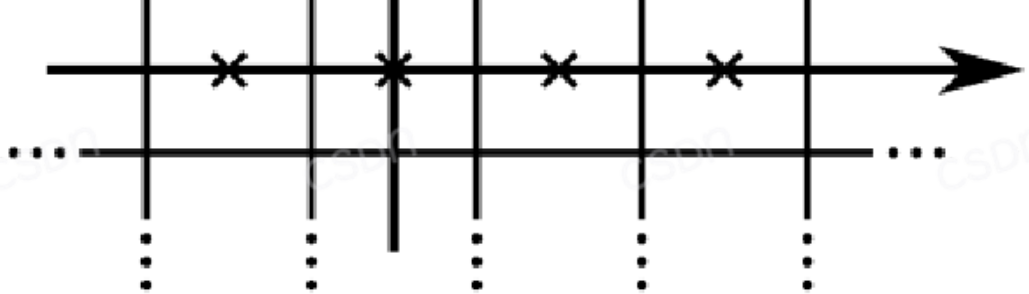
如何使格子的评分是其子格子的评分上界？

一个位姿的评分是用该位姿下所有激光束（beams）的命中点（hit point）的评分相加得到的。

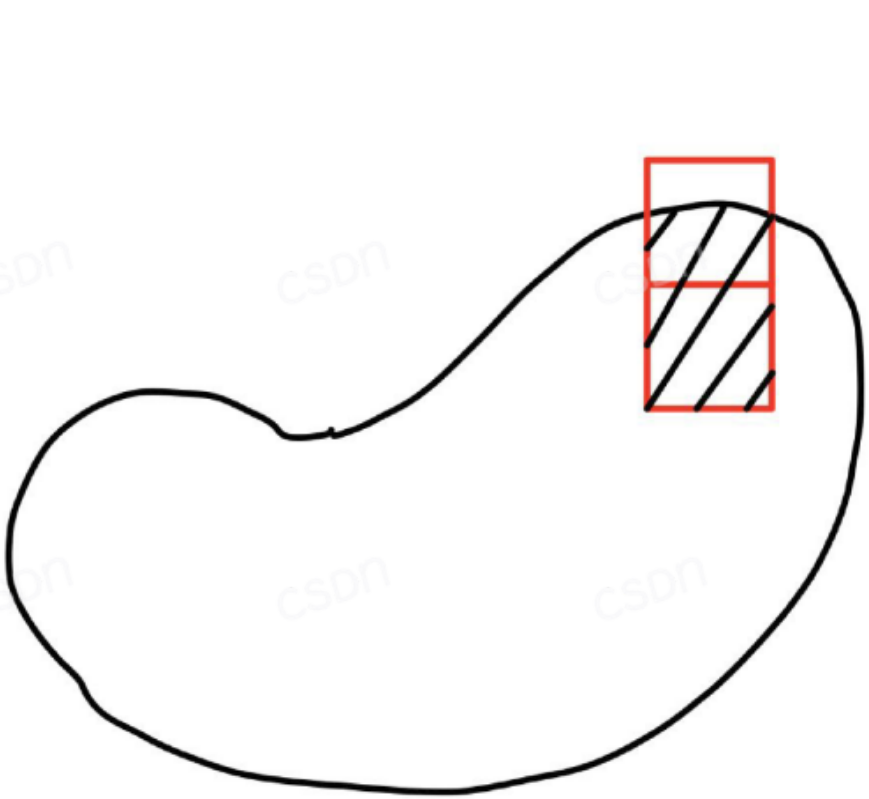
$$\xi^{\star} = \underset{\xi \in \mathcal{W}}{\operatorname{argmax}} \sum_{k=1}^{K} M_{\text{nearest}}(T_{\xi} h_k), \qquad \text{(BBS)}$$

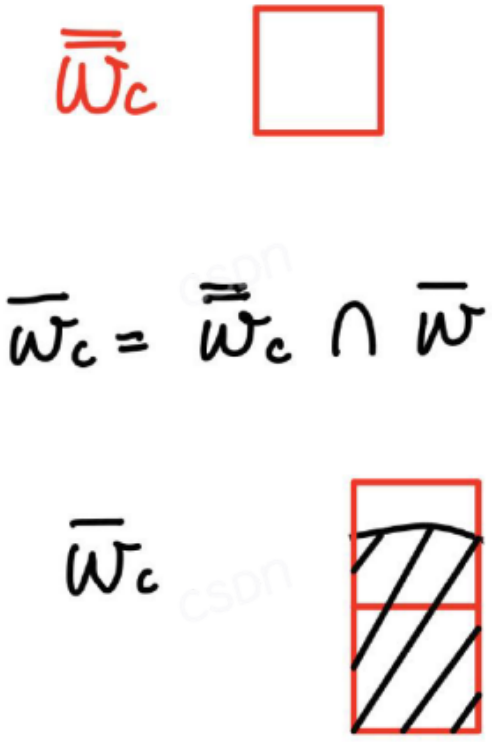






暴力搜索：用三层for循环，把每一个位姿都拿出来算一遍BBS评分，找到最大值。

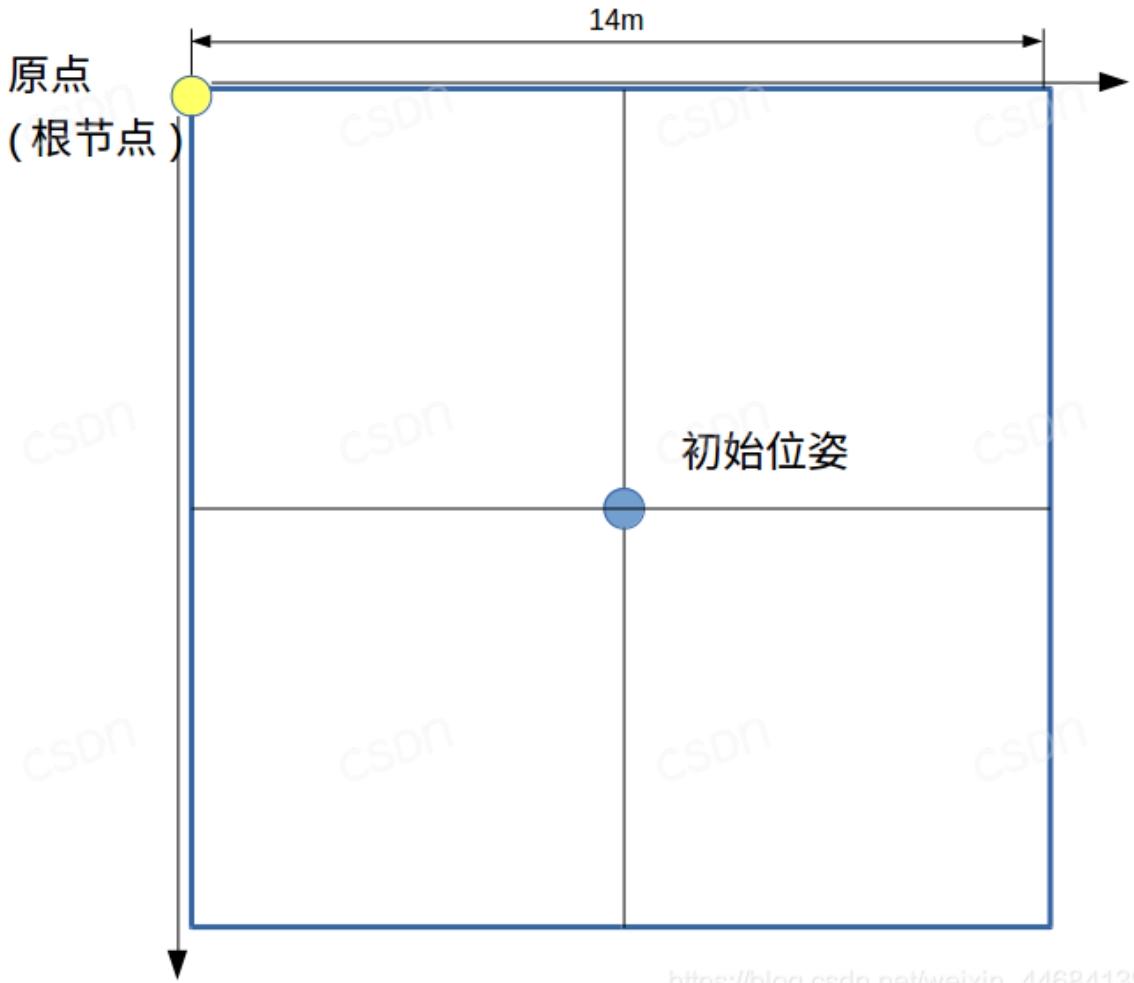




$$score(c) = \sum_{k=1}^{K} \max_{j \in \overline{\overline{\mathcal{W}}}_c} M_{\text{nearest}}(T_{\xi_j} h_k)$$

$$\geq \sum_{k=1}^{K} \max_{j \in \overline{\mathcal{W}}_c} M_{\text{nearest}}(T_{\xi_j} h_k)$$

$$\geq \max_{j \in \overline{\mathcal{W}}_c} \sum_{k=1}^{K} M_{\text{nearest}}(T_{\xi_j} h_k).$$



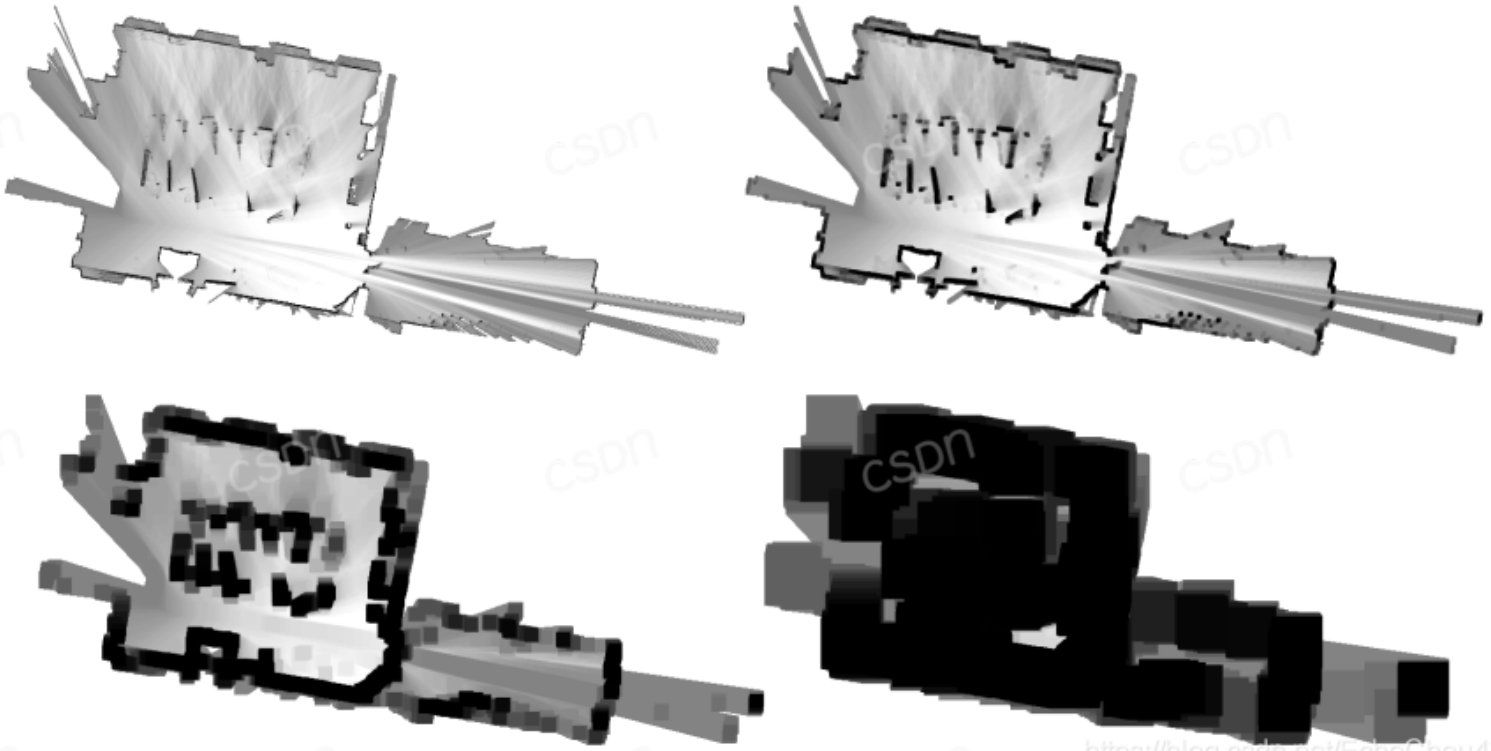
最大值的和：对每一根激光束，都取一个搜索框范围内与他对应的位姿，使这根激光束的评分Mnearest最大，把每一个评分累加，求和。这个就是中间节点的评分上界，即位姿评分的上界，为每个命中点（hit point）的评分的上界之和。

和的最大值：对每一个位姿，计算该位姿下，所有激光束的所有激光束（beams）的命中点的评分，累加求和，在这些和中找最大值（这也就是BBS公式，代表最细分辨率上的得分，分枝定界并没有真正的去求解所有枚举情况的目标函数BBS值）

采用预计算网格方法：通过构造多分辨率地图，把地图中每个格子的概率用其附近区域内的最大概率代替，使得每一个节点的评分可以在对应分辨率地图中查询得到。不同分辨率的地图事先计算好。越低分辨率的地图，在越大的范围里求最大值，格子中的概率越接近1。

$$score(c) = \sum_{k=1}^{K} M_{\text{precomp}}^{c_h}(T_{\xi_c} h_k),$$

$$M_{\text{precomp}}^{h}(x, y) = \max_{\substack{x' \in [x, x + r(2^h - 1)] \\ y' \in [y, y + r(2^h - 1)]}} M_{\text{nearest}}(x', y')$$



总结

分枝定界进行分枝的过程，是不断提高搜索精度的过程。

整个分枝树的最底层的所有枚举情况，便是最高搜索精度的枚举集合，便是暴力匹配搜索范围的全部整搜索空间。