

怎么做分支

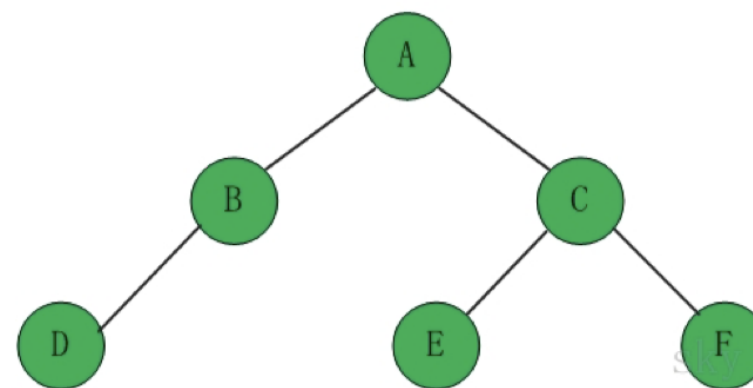
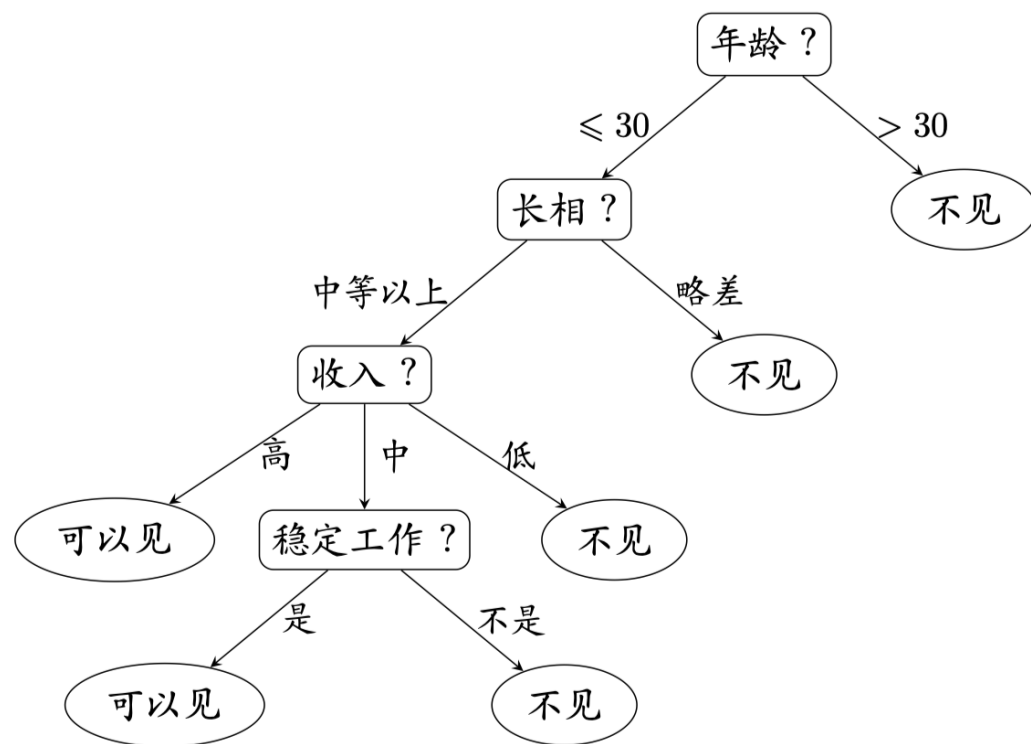
zhu.qw@qq.com

获取最新: <https://github.com/oak97/branch-tutorial>

Part1 Branch

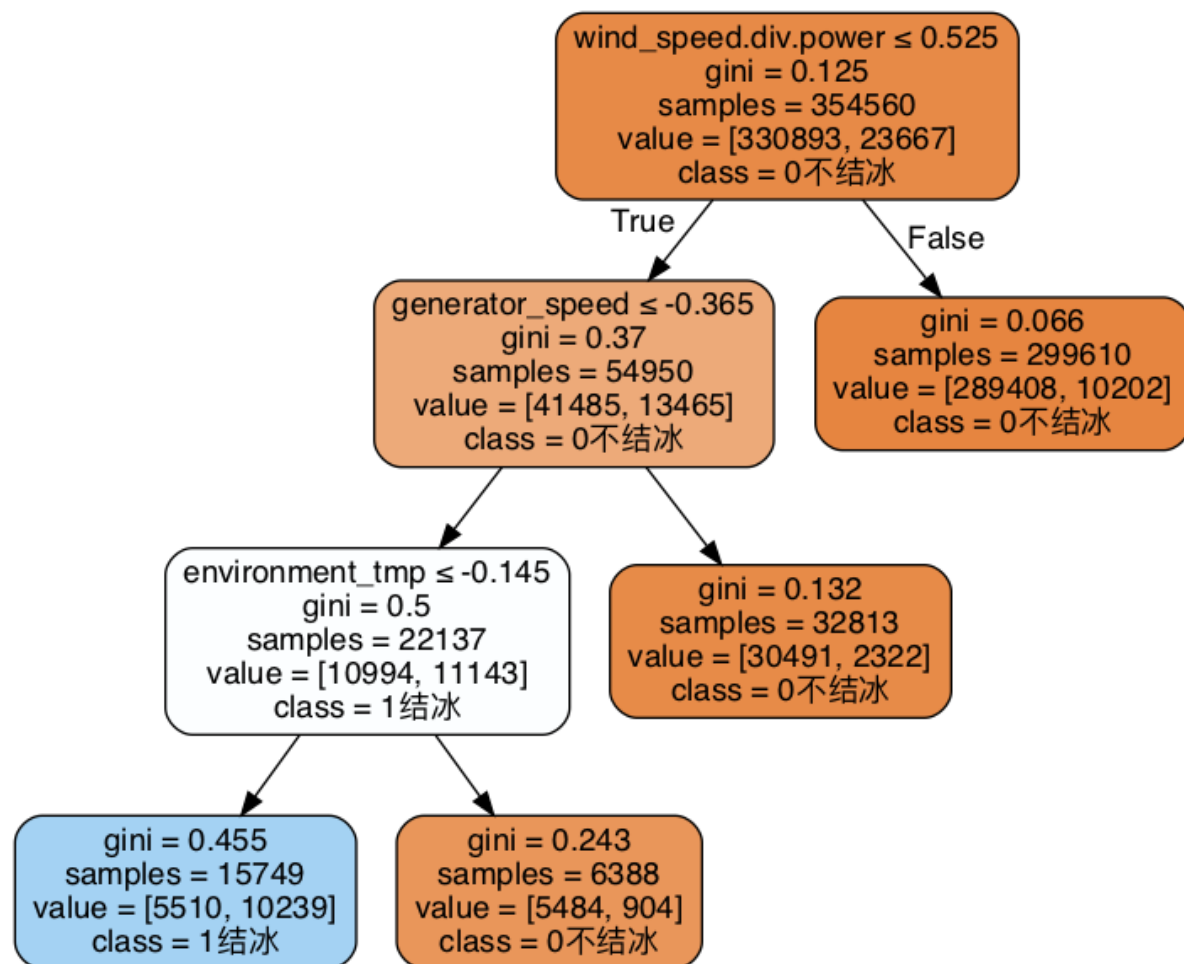
- 树结构的分支
- 决策树分支
- 寻路分支

树结构的分支



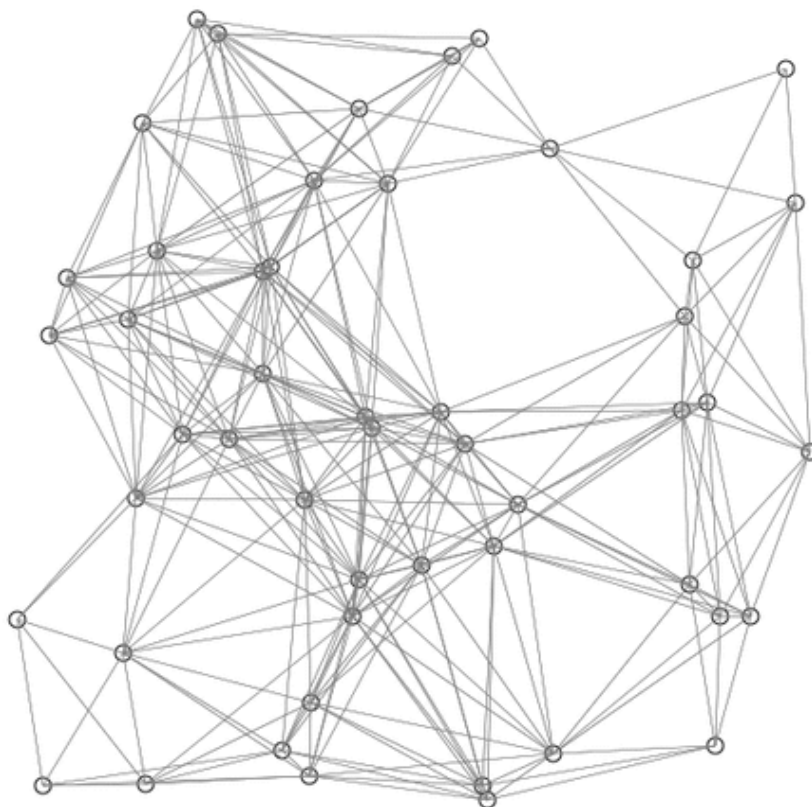
决策树分支

- 特征选择
 - 哪个特征
 - 哪个值
- 决策树的生成
- 决策树的修剪



寻路分支

- 搜索
 - 下一步走哪个点



Part2 Bound

- branch为什么要加入bound: 剪枝
- lb和ub

有信息搜索

- 最短路径 (最小化问题)
- heuristic function
当前节点到终点的想象中的最短距离 (直线、曼哈顿...)

- evaluation function
 - $h() + \text{起点到当前节点的真实距离}$
 - **当前节点的下界 lower bound**
 $\leq \text{真实总距离 (未知)}$

- **upper bound**
 $= \text{当前最好距离} \geq \text{最优距离 (未知)}$
 - 更新

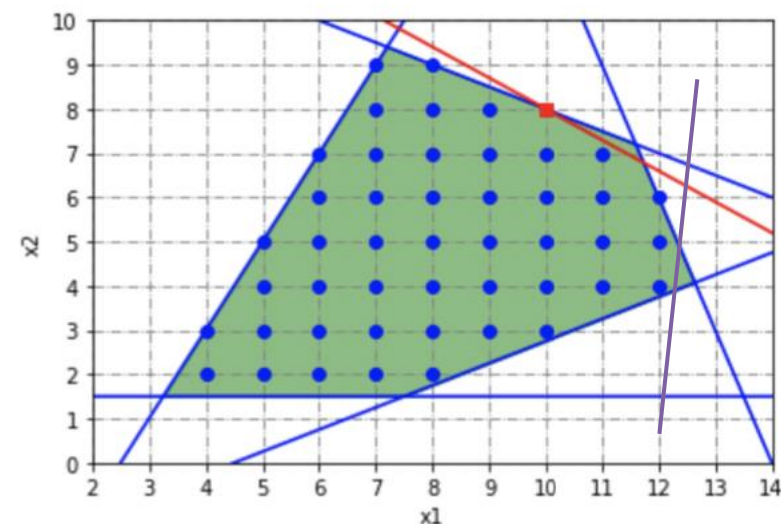
- **剪枝**: $\text{真实总距离 (未知)} \geq \text{lb} > \text{ub} = \text{当前最好距离} \geq \text{最优距离 (未知)}$
即 $\text{真实总距离 (未知)} > \text{当前最好距离} \geq \text{最优距离 (未知)}$
所以 当前节点不可能是最优解, 也不可能是我们最终采取的解

Classes of Search

Blind (uninformed)	Depth-First Breadth-First Iterative-Deepening	Systematic exploration of whole tree until the goal is found.
Heuristic (informed)	Hill-Climbing Best-First Beam	Uses heuristic measure of goodness of a node, e.g. estimated distance to goal.
Optimal (informed)	Branch&Bound A*	Uses path "length" measure. Finds "shortest" path. A* also uses heuristic

Part3 (M)IP

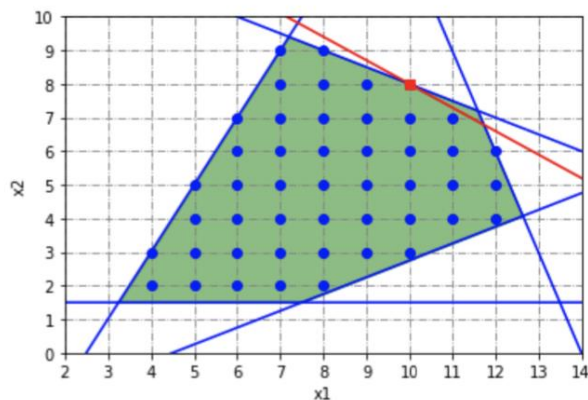
- LP-based B&B: LP作为IP的下界
- 松弛后的LP恰是原IP的下界
 - 不用像前面最短路问题尝试各种heuristic function
 - 也不用不同问题手工定制
- 松弛后的LP无可行解: 剪枝
- LP的可行解如果也是IP可行: 更新上界, 剪枝 (不用继续分支)



IP例子 MAX问题

原IP问题

$$\begin{aligned} \max \quad & 7x_1 + 10x_2 \\ \text{s.t.} \quad & -x_2 \leq -1.5 \\ & x_1 + 2x_2 \leq 26 \\ & 2x_1 - 4x_2 \leq 9 \\ & -2x_1 + x_2 \leq -5 \\ & 3x_1 + x_2 \leq 42 \\ & \underline{x_1, x_2 \in \mathbb{Z}_+} \end{aligned}$$

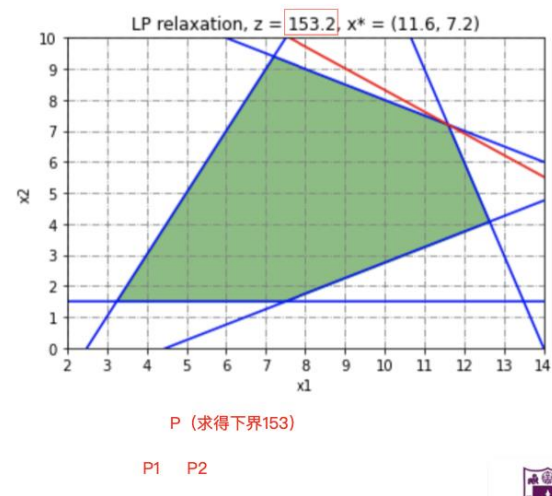


原IP问题对应的LP: P

Root node 求解线性规划松弛

$$z = \max_{(x_1, x_2) \in P} 7x_1 + 10x_2$$

$$P = \{(x_1, x_2) \in \mathbb{R}_+^2 : \begin{aligned} & -x_2 \leq -1.5, \\ & x_1 + 2x_2 \leq 26, \\ & 2x_1 - 4x_2 \leq 9, \\ & -2x_1 + x_2 \leq -5, \\ & 3x_1 + x_2 \leq 42 \end{aligned}\}$$



求出P的最优解 x^* 和最优值 z ，没有事先求出IP可行解作为下界，这里的 x^* 不是IP可行，所以也不作为下界，P的上界是153.2

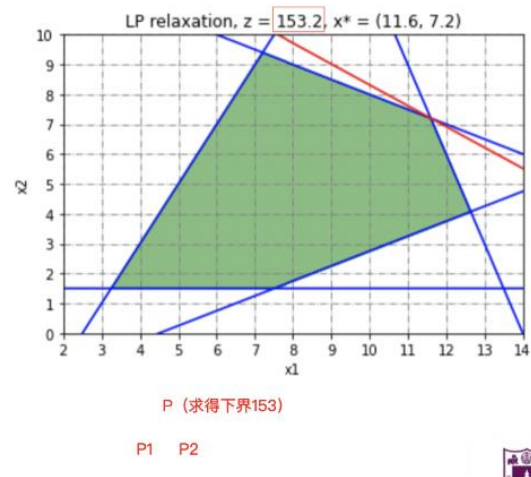
IP例子 MAX问题

原IP问题对应的LP : P

Root node 求解线性规划松弛

$$z = \max_{(x_1, x_2) \in P} 7x_1 + 10x_2$$

$$P = \{(x_1, x_2) \in \mathbb{R}_+^2 : \begin{aligned} &-x_2 \leq -1.5, \\ &x_1 + 2x_2 \leq 26, \\ &2x_1 - 4x_2 \leq 9, \\ &-2x_1 + x_2 \leq -5, \\ &3x_1 + x_2 \leq 42 \end{aligned}\}$$



求出P的最优解 x^* 和最优值 z ，没有事先求出IP可行解作为下界，这里的 x^* 不是IP可行，所以也不作为下界，P的上界是153.2

P分支得到P1和P2：对P的 $x_1=11.6$ 进行分支

第一次分支: P **Root node**

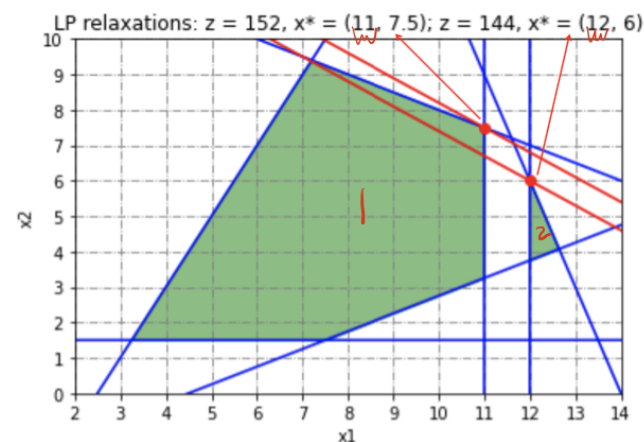
- $x_1 \leq 11$ 和 $x_1 \geq 12$

Child node 1 问题1

- $z = \max_{(x_1, x_2) \in P_1} 7x_1 + 10x_2$
- $P_1 = P \cap \{x_1 \leq 11\}$

Child node 2

- $z = \max_{(x_1, x_2) \in P_2} 7x_1 + 10x_2$
- $P_2 = P \cap \{x_1 \geq 12\}$



分别求出P1和P2的最优解 x^* 和最优值 z ，P1上界为152，P2上界为144，且P2的 x^* 是IP可行，所以更新下界为144，P2不分支生成节点了

IP例子 MAX问题

P分支得到P1和P2：对P的 $x_1=11.6$ 进行分支

第一次分支: P Root node

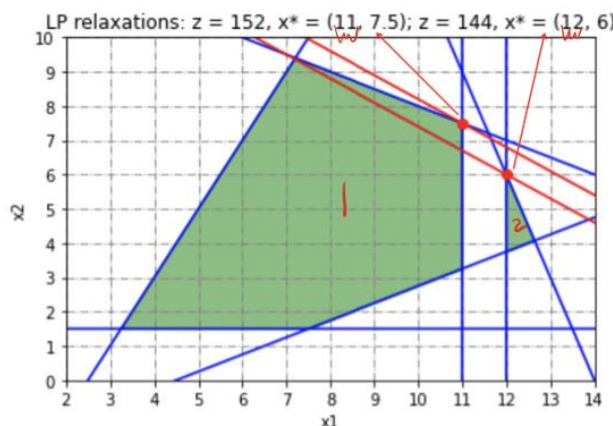
- $x_1 \leq 11$ 和 $x_1 \geq 12$

Child node 1 问题1

- $z = \max_{(x_1, x_2) \in P_1} 7x_1 + 10x_2$
- $P_1 = P \cap \{x_1 \leq 11\}$

Child node 2

- $z = \max_{(x_1, x_2) \in P_2} 7x_1 + 10x_2$
- $P_2 = P \cap \{x_1 \geq 12\}$



P1分支得到P3和P4：对P1的 $x_2=11.6$ 进行分支

第二次分支: P_1 child node 1

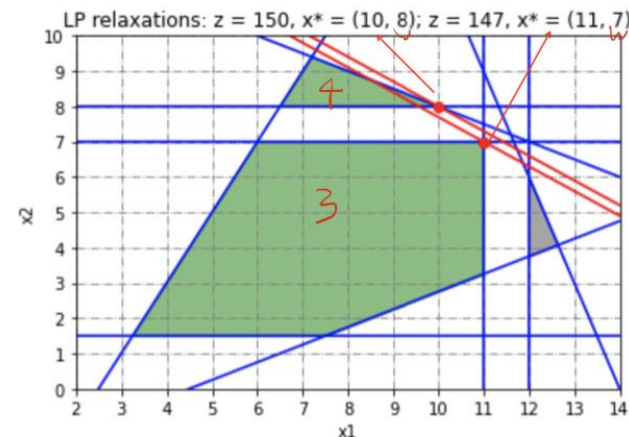
- $x_2 \leq 7$ 和 $x_2 \geq 8$

Child node 3

- $z = \max_{(x_1, x_2) \in P_3} 7x_1 + 10x_2$
- $P_3 = P_1 \cap \{x_2 \leq 7\}$

Child node 4

- $z = \max_{(x_1, x_2) \in P_4} 7x_1 + 10x_2$
- $P_4 = P_1 \cap \{x_2 \geq 8\}$




分别求出P3和P4的最优解和最优值，P3上界为147，P3的 x^* 是IP可行，并且 $147 > 144$ ，所以更新下界为147。P4上界为150，P4的 x^* 也IP可行，并且 $150 > 147$ ，所以更新下界为150，不再分支

💡 原IP问题的最优值就是150

Branch-and-bound

- 性能因素：实际求解node数目 × 每个node求解效率
- 尽量减少实际求解node数目
- 提升每个node求解效率

减少实际求解node数目

- 更好的上界：启发式算法
- 更好的下界
 - valid inequalities
 - column generation
- 分支策略
- node搜索策略 

剪枝条件：真实距离（未知） $\geq lb > ub$ = 当前最好距离 \geq 最优距离（未知）
lb越大，ub越小，剪枝条件成立越多

改善单node求解效率

- 更好的formulation
- 去除冗余的约束，变量赋值，presolve
- 高效的松弛问题求解

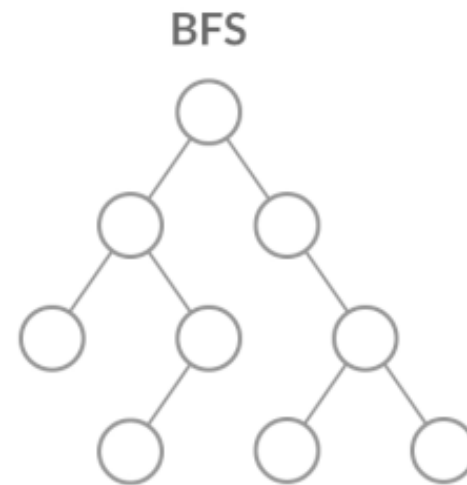
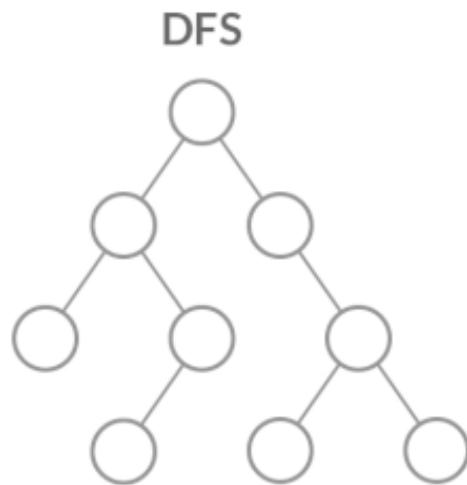
分支策略

- 分支策略选取：不有效的分支会导致子问题很多
- Random Branching：随机选取非整数的变量
- Most Infeasible Branching：小数部分越接近0.5，就越远离整数
- Pseudo Cost Branching：记录之前选择各个变量作为分支变量时目标函数的变化，然后根据过去目标函数值的变化预测此时选择该变量将导致的目标函数变化，选择预计使目标函数变化最大的变量
- Strong Branching：在实际分支之前测试哪个变量对目标函数的改进最大
- Reliability Branching
- Local Branching

Achterberg T, Koch T, Martin A. Branching rules revisited[J]. Operations Research Letters, 2005, 33(1): 42-54.
Fischetti M, Lodi A. Local branching[J]. Mathematical programming, 2003, 98(1-3): 23-47.

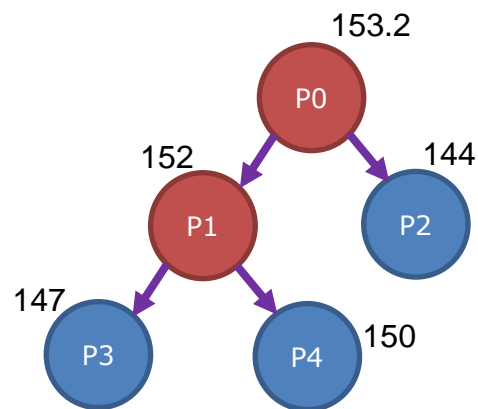
节点搜索（选择）策略

- 深度优先
- 广度优先：最短层数的完整解
- 最佳优先
-



- 三种实现的代码讲解 对应前面的图示例子

<https://github.com/oak97/branch-tutorial/tree/master/src/search/strategy>



深度优先	0 1 3 4 2
广度优先	0 1 2 3 4
最佳优先	0 1 4 3 2

深度优先：栈

1. 首先将根节点放入`stack`中。
2. 从`stack`中取出第一个节点，并检验它是否为目标。
如果找到目标，则结束搜寻并回传结果。
否则将它某一个尚未检验过的直接子节点加入`stack`中。
3. 重复步骤2。
4. 如果不存在未检测过的直接子节点。
将上一级节点加入`stack`中。
重复步骤2。
5. 重复步骤4。
6. 若`stack`为空，表示整张图都检查过了——亦即图中没有欲搜寻的目标。结束搜寻并回传“找不到目标”。

广度优先：队列

1. 首先将根节点放入队列中。
2. 从队列中取出第一个节点，并检验它是否为目标。
 - 如果找到目标，则结束搜索并回传结果。
 - 否则将它所有尚未检验过的直接子节点加入队列中。
3. 若队列为空，表示整张图都检查过了——亦即图中没有欲搜索的目标。结束搜索并回传“找不到目标”。
4. 重复步骤2。

最佳优先：优先队列

Greedy BFS [\[edit\]](#)

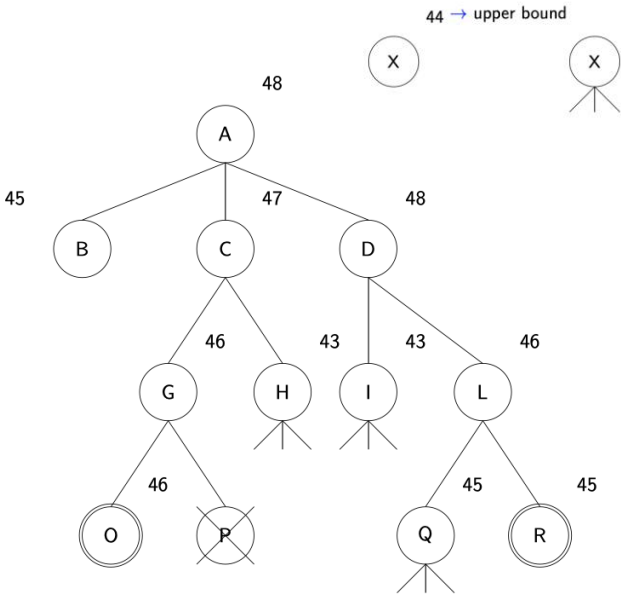
Using a [greedy algorithm](#), expand the first successor of the parent. After a successor is generated:^{[\[4\]](#)}

1. If the successor's heuristic is better than its parent, the successor is set at the front of the queue (with the parent reinserted directly behind it), and the loop restarts.
2. Else, the successor is inserted into the queue (in a location determined by its heuristic value). The procedure will evaluate the remaining successors (if any) of the parent.

最佳优先 和 深度优先

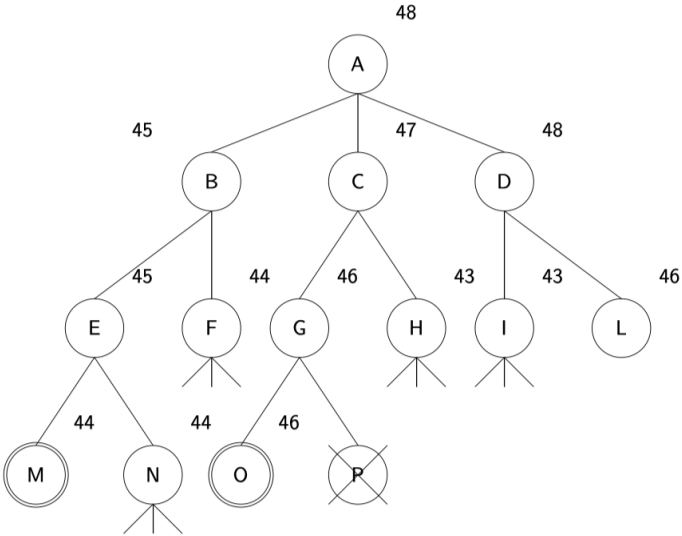
maximization problem

	Division	\mathcal{L}	Chosen
1.		A	A
2.	B,C,D	B,C,D	D
3.	I,L	B,C,I,L	C
4.	G,H	B,G,H,I,L	L
5.	Q,R	G	G
6.	O,P	—	



List of nodes explored:

A - B - E - M - (E) - N -
(E) - (B) - F - (B) - (A) -
C - G - O - (G) - (C) - H -
(C) - (A) - D - I - (D) - L



Best First

- The next problem to be analyzed is selected by choosing the subproblem with the best bound (lowest lower bound, if we are minimizing)
- Its generally minimize the size of the enumeration tree, thus minimizing the overall solution time
- Drawbacks:
 - Doesn't find feasible solutions quickly
 - Implementation: memory usage and node setup costs

常用

Depth First

- The next problem to be analyzed is the *deepest* node
- This avoids most of the problems with best first:
 - Implementation: the number of candidate nodes is minimized (saving memory), and node set-up costs are minimized
 - Feasible solutions are found more quickly
- Drawbacks: usually leads to a larger enumeration tree

容易更新上界：因为多次不同变量分支，容易得到IP可行的解

Summary of search strategies

Strategy	Selection from Frontier	Path found	Space
Breadth-first	First node added	Fewest arcs	Exponential
Depth-first	Last node added	No	Linear
Iterative deepening	—	Fewest arcs	Linear
Greedy best-first	Minimal $h(p)$	No	Exponential
Lowest-cost-first	Minimal cost (p)	Least cost	Exponential
A^*	Minimal cost $(p) + h(p)$	Least cost	Exponential
IDA*	—	Least cost	Linear

“Path found” refers to guarantees about the path found (for graphs with finite branching factor and arc costs bounded above zero). The algorithms that guarantee to find a path with fewest arcs or least cost are complete. “No” means that it is not guaranteed to find a path in infinite graphs. Both depth-first search and greedy best-first search can fail to find a solution on finite graphs with cycles unless cycle pruning or multiple-path pruning is used.

Space refers to the space complexity, which is either “Linear” in the maximum number of arcs in a path expanded before a solution is found or “Exponential” in the number of arcs in a path expanded before a solution is found.

Iterative deepening is not an instance of the generic search algorithm, and so the iterative deepening methods do not have an entry for the selection from the frontier.

Figure 3.11: Summary of search strategies

Part4 VRP

- 车辆流模型下的单纯B&B
- 集合覆盖模型下的B&P&C的Branch部分
 - 用列生成求解LP
 - 定价子问题求解
 - Cut

- 代码讲解

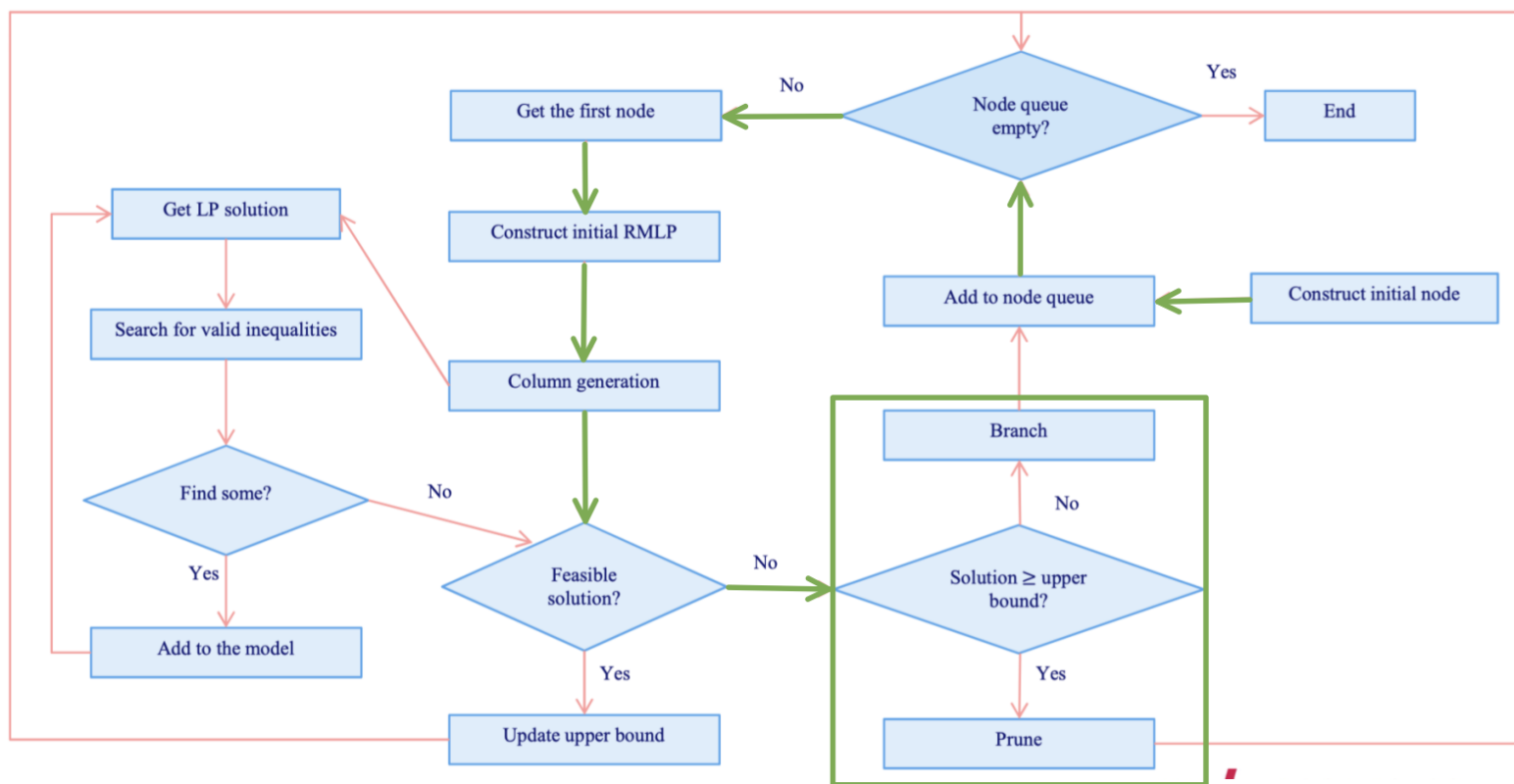
代码见数据魔术师公众号

运筹学教学|分支定界法解带时间窗的车辆路径规划问题（附代码及详细注释）

<https://mp.weixin.qq.com/s/AwEYJyITeAcwC2W5jIjQw>

- 建模是流模型，即决策变量是 x_{ij} ，先在LP中将决策变量松弛成 $[0,1]$ ，然后对非整数的 x_{ij} 进行分支，直至全部都变为整数变量

- 建模变为集合覆盖模型
- 绿色框



- 整体框架: `Run.main()`
- 分支框架: `Tree.solve()`
 - 最佳优先
 - 深度优先
 - 广度优先
- 分支策略: `Branch.branch()`

分支策略 6.3 节

- 根据使用的车辆总数进行分支
- 根据访问每个客户的车辆数进行分支
- 根据两个客户之间的车辆流总和进行分支
- 根据连续三个客户之间的车辆流总和进行分支
- 根据每个时间窗被选择的次数进行分支（用分支来处理难约束）

根据使用的车辆总数进行分支

- 代码讲解
- `Branch.fraction_vehicle()`
- `Branch.branch_vehicle()`

根据每个时间窗被选择的次数进行分支

- 代码讲解
- 用分支来处理难约束
- `Branch.fraction_window()`
- `Branch.branch_window()`

Part5 Take-Away

- branch
- bound
- LP-based B&B
- VRP: VRPTW单纯B&B => 论文的B&P&C
- <https://github.com/oak97/branch-tutorial>

Thanks