PKU-04833060
算法设计与分析『实验班』

第6讲 回溯与分支限界 (2/2)

罗国杰

gluo@pku.edu.cn

2025年春季学期

最大团问题

问题：给定无向图$G=<V, E>$, 求G中的最大团.

相关知识：

无向图$G = <V, E>$,

G的子图：$G' = <V', E'>$, 其中$V' \subseteq V$, $E' \subseteq E$,

G的补图：$\check{G} = <V, E'>$, E'是E关于完全图边集的补集

G中的团：G 的完全子图

G 的点独立集：G 的顶点子集A，且$\forall u,v \in A, (u,v) \notin E$.

最大团：顶点数最多的团

最大点独立集：顶点数最多的点独立集

命题：U是G 的最大团当且仅当U是$\check{G}$ 的最大点独立集

最大团问题：算法设计

结点$<x_1, x_2, \ldots, x_k>$的含义：

已检索 k 个顶点，其中 $x_i=1$ 对应的顶点在当前的团内

搜索树为子集树

约束条件：该顶点与当前团内每个顶点都有边相连

界：当前图中已检索到的极大团的顶点数

代价函数：目前的团扩张为极大团的顶点数上界

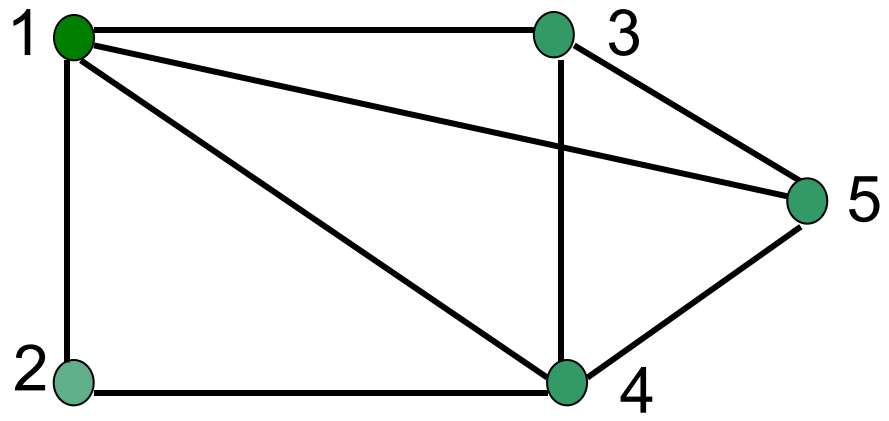
$$F = C_n + n - k$$

其中 C_n 为目前团的顶点数（初始为0），

k 为结点层数

时间：$O(n2^n)$

最大团的实例



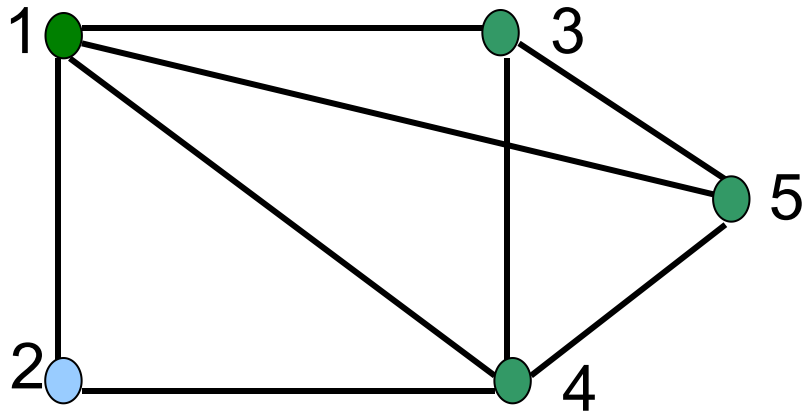
顶点编号顺序为 1, 2, 3, 4, 5,

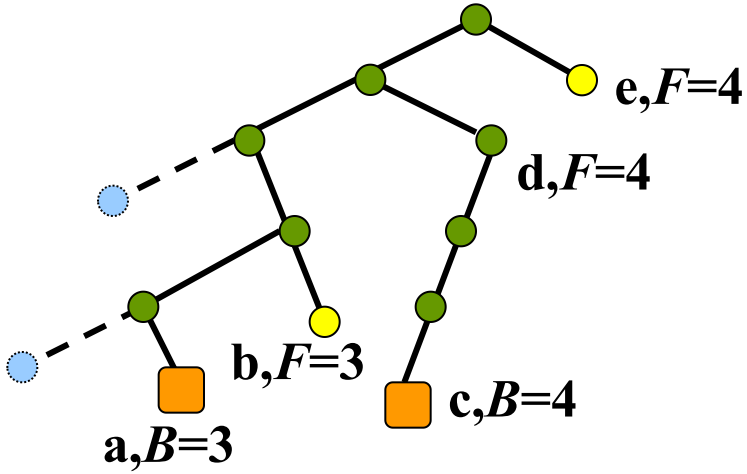
对应 x_1, x_2, x_3, x_4, x_5, x_i=1 当且仅当 i 在团内

分支规定左子树为1，右子树为0.

B 为界，F 为代价函数值.

最大团问题：实例求解





a：得第一个极大团 { 1, 2, 4 }, 顶点数为3, 界为3；
b：代价函数值 $F = 3$, 回溯；
c：得第二个极大团{1, 3, 4, 5}, 顶点数为4, 修改界为4；
d：不必搜索其它分支, 因为$F = 4$, 不超过界；
e：$F = 4$, 不必搜索.
最大团为 {1, 3, 4, 5}, 顶点数为 4.

最大团问题：改进的扩张策略和代价函数

- 原策略
  - 以任意顺序搜索顶点
  - 以 $F = C_n + (n-k)$ 为代价函数
- 改进策略
  - 以度数降序或升序搜索顶点
    - 降序：更快找到最大团、但可能更慢才能证明最优性
  - 以 $F = C_n + color$ 为代价函数
    - color 为用贪心法给余下 n-k 个顶点的染色数
    - 余下 n-k 顶点子图的最大团，顶点数 ≤ color
    - |原图最大团| ≤ |k顶点子图最大团| + |n-k顶点子图最大团| ≤ C_n + color

David R Wood, "An algorithm for finding a maximum clique in a graph," Operations Research Letters, vol. 21, no. 5, 1997, pp. 211-217.

货郎问题

问题：给定n个城市集合$C=\{c_1,c_2,\ldots,c_n\}$, 从一个城市到另一个城市的距离 d_{ij} 为正整数，求一条最短且每个城市恰好经过一次的巡回路线.

货郎问题的类型：有向图、无向图.

设巡回路线从1开始，
解向量为$<i_1, i_2, \ldots, i_{n-1}>$,
其中 $i_1, i_2, \ldots, i_{n-1}$为$\{ 2, 3, \ldots, n \}$的排列.
搜索空间为排列树，结点$<i_1, i_2, \ldots, i_k>$ 表示得到 k 步路线

货郎问题：算法设计 (1/2)

约束条件：令$B=\{i_1, i_2, \ldots, i_k\}$, 则

$$i_{k+1} \in \{2, \ldots, n\} - B$$

界：当前得到的最短巡回路线长度

代价函数：设顶点c_i出发的最短边长度为l_i，d_j为选定巡回路线中第j段的长度，则

$$L=\sum_{j=1}^{k-1} d_j + l_{i_k} + \sum_{i_j \notin B} l_{i_j}$$

为部分巡回路线扩张成全程巡回路线的长度下界

时间$O(n!)$：计算$O((n-1)!)$次，代价函数计算$O(n)$

货郎问题：算法设计 (2/2)

代价函数1：设顶点 c_i 出发的最短边长度为 l_i，d_j 为选定巡回路线中第 j 段的长度，则

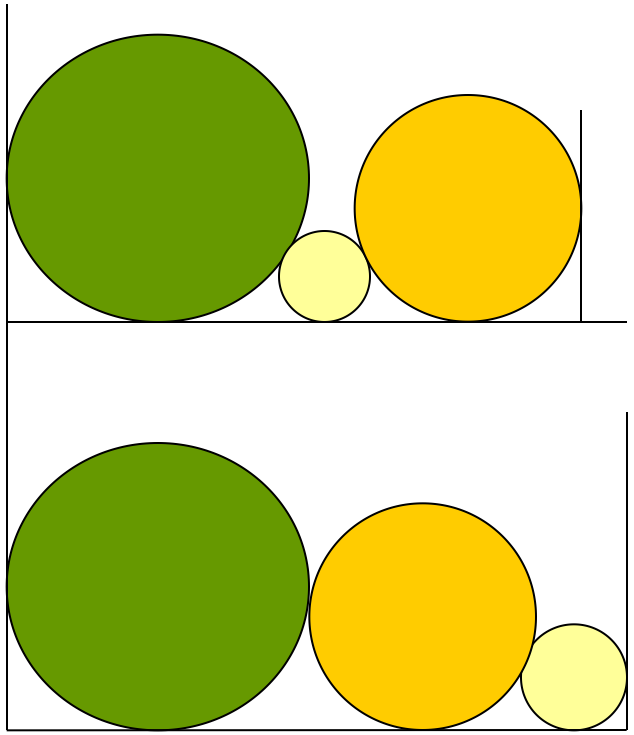
$$L=\sum_{j=1}^{k-1}d_j+l_{i_k}+\sum_{i_j\notin B}l_{i_j}$$

代价函数2：设顶点 c_i 出发的最短边长度为 l_i、次短边长度为l'_i，d_j 为选定巡回路线中第 j 段的长度，则

$$L=\sum_{j=1}^{k-1}d_j+\sum_{i_j\notin B}(l_{i_j}+l'_{i_j})/2$$

圆排列问题

问题：给定n个圆的半径序列, 将各圆与矩形底边相切排列, 求具有最小长度 l_n 的圆的排列顺序.



解为$<i_1, i_2, \dots, i_n>$为$1, 2, \dots, n$的排列，解空间为排列树.

部分解向量 $<i_1, i_2, \dots, i_k>$：表示前 k 个圆已排好. 令$B=\{ i_1, i_2, \dots, i_k \}$,下一个圆选择 i_{k+1}.

约束条件：$i_{k+1} \in \{1, 2, \dots, n\} - B$

界：当前得到的最小圆排列长度

圆排列问题：代价函数符号说明

k：算法完成第 k 步，已经选择了第1至k 个圆

r_k：第 k 个圆的半径

d_k：第 k–1 个圆到第 k 个圆的圆心水平距离，k>1

x_k：第 k 个圆的圆心坐标，规定 x_1=0,

l_k：第 1至 k 个圆的排列长度

L_k：放好 1至k 个圆以后，对应结点的代价函数值

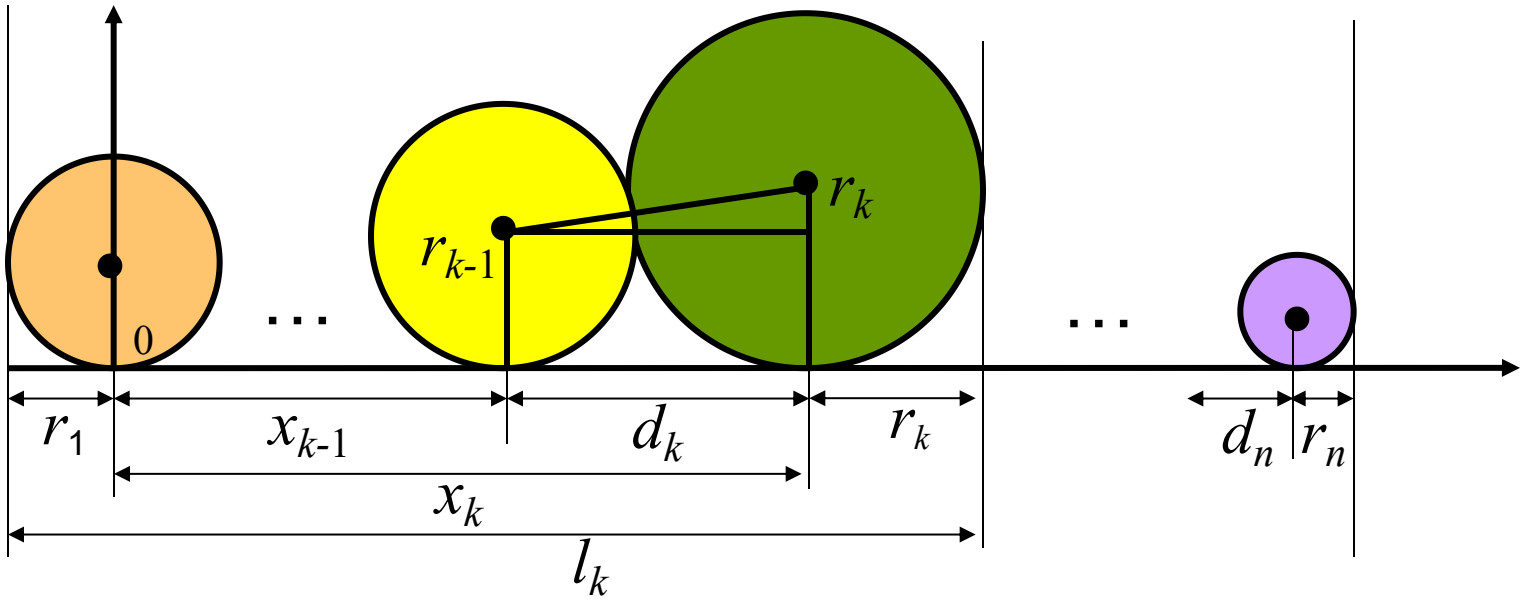
满足 $L_k \le l^*_n \le l_n$

圆排列问题：有关量的计算

$$d_k = \sqrt{(r_{k-1}+r_k)^2-(r_{k-1}-r_k)^2} = 2\sqrt{r_{k-1}r_k}$$

$$x_k \geq x_{k-1}+d_k, \qquad l_k \geq x_k + r_k + r_1$$

$$\begin{aligned} l_n &\geq x_k + d_{k+1} + d_{k+2} + \ldots + d_n + r_n + r_1 \\ &\geq x_k + 2\sqrt{r_k r_{k+1}} + 2\sqrt{r_{k+1}r_{k+2}} + \ldots + 2\sqrt{r_{n-1}r_n} + r_n + r_1 \end{aligned}$$



圆排列问题：代价函数

排列长度是l_n，L是代价函数：

$$
\begin{aligned}
l_n &\geq x_k + 2\sqrt{r_k r_{k+1}} + 2\sqrt{r_{k+1} r_{k+2}} + \ldots + 2\sqrt{r_{n-1} r_n} + r_n + r_1 \\
&\geq x_k + 2(n-k)r + r + r_1 \\
L &= x_k + (2n - 2k + 1)r + r_1 \\
r &= \min(r_{i_j}, r_k) \quad i_j \in \{1, 2, \ldots, n\} - B
\end{aligned}
$$

$$B = \{ i_1, i_2, \ldots, i_k \},$$

时间：$O(n\ n!)=O((n+1)!)$

圆排列问题：实例计算过程

$R = \{1, 1, 2, 2, 3, 5\}$

取排列 <1, 2, 3, 4, 5, 6>,

半径排列为：1, 1, 2, 2, 3, 5，结果见下表和下图

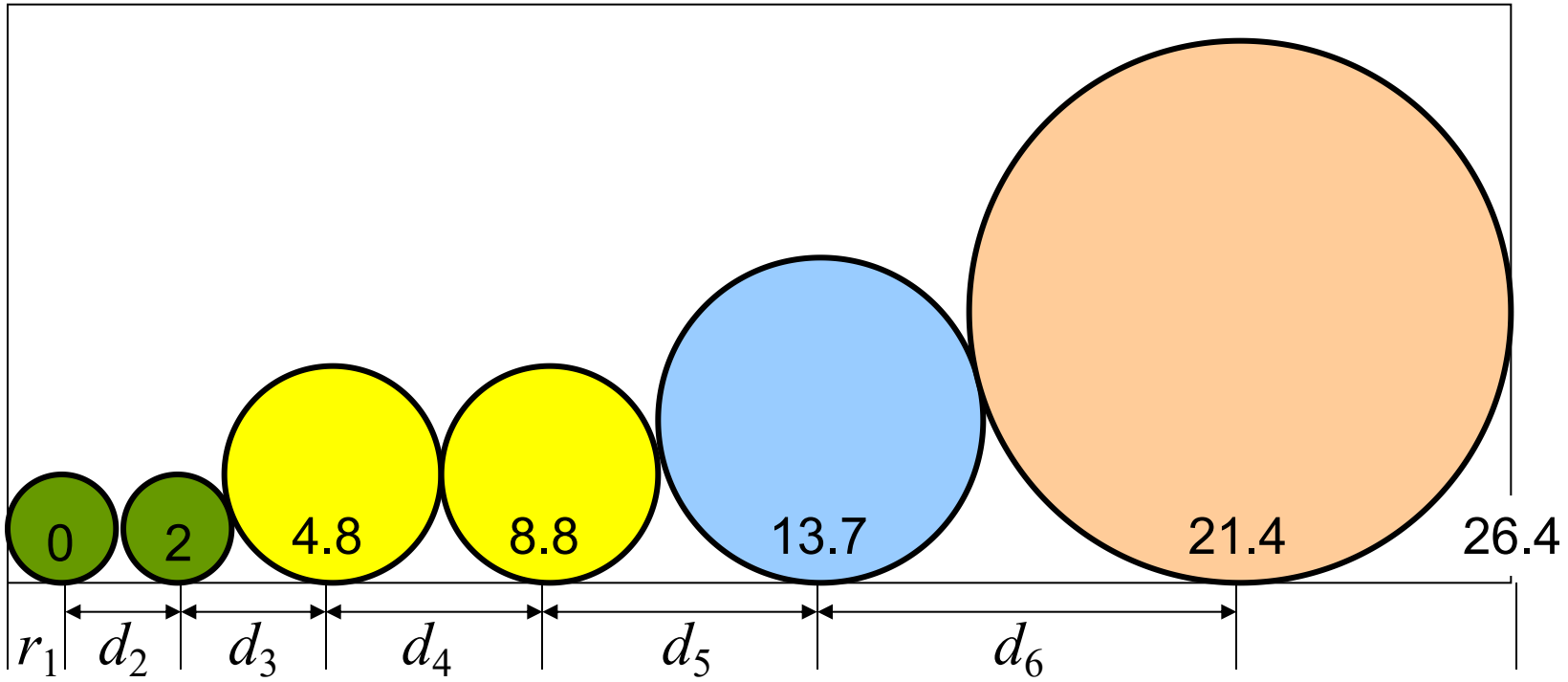
k	r_k	d_k	x_k	l_k	L_k
1	1	0	0	2	12
2	1	2	2	4	12
3	2	2.8	4.8	7.8	19.8
4	2	4	8.8	11.8	19.8
5	3	4.9	13.7	17.7	23.7
6	5	7.7	21.4	27.4	27.4

圆排列问题：实例图示

$R = \{1, 1, 2, 2, 3, 5\}$

取排列 <1, 2, 3, 4, 5, 6>, 半径排列为: 1, 1, 2, 2, 3, 5，

最短长度 $l_6 = 27.4$



连续邮资问题

问题：给定n种不同面值的邮票，每个信封至多m张， 试给出邮票的最佳设计，使得从1开始，增量为1的连续邮资区间达到最大？

实例：n=5，m=4，
面值 X_1=<1,3,11,15,32>，邮资连续区间为{ 1, 2, …,70 }
面值 X_2=<1,6,10,20,30>，邮资连续区间为{1, 2, 3, 4}

可行解：$<x_1, x_2, \ldots, x_n>$，$x_1=1$，$x_1 < x_2 < \ldots < x_n$
约束条件：在结点$<x_1, x_2, \ldots, x_i>$处，邮资最大连续区间为$\{ 1, \ldots, r_i \}$，x_{i+1}的取值范围是$\{ x_i+1, \ldots, r_i+1\}$

连续邮资问题：r_i 的计算

$y_i(j)$：用至多 m 张面值 x_i 的邮票加上 $x_1, x_2, \ldots, x_{i-1}$ 面值的邮票贴 j 邮资时的最少邮票数，则

$$y_i(j) = \min_{1 \le t \le m} \{t + y_{i-1}(j - t\,x_i)\}$$

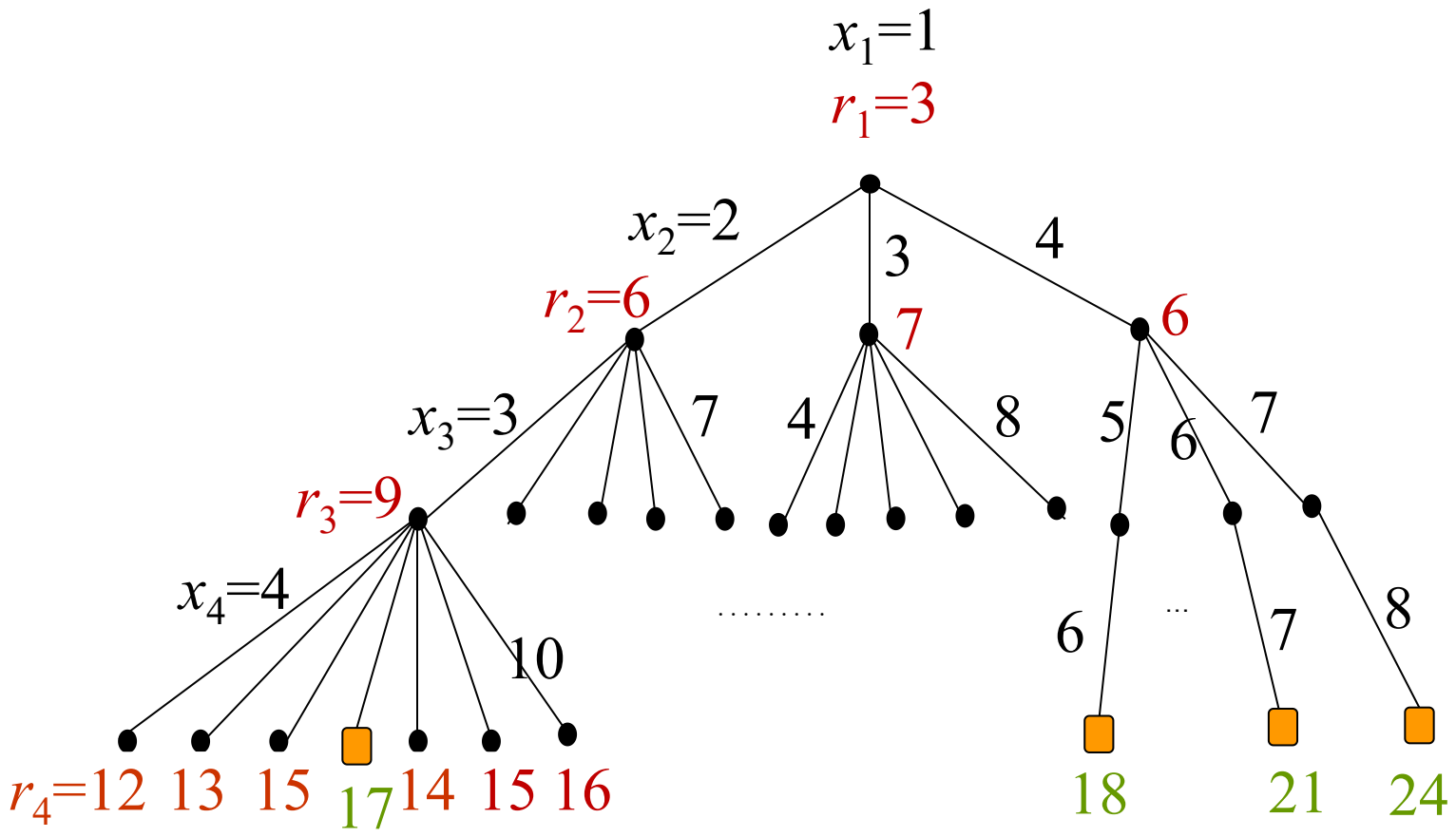
$$y_1(j) = j$$

$$r_i = \min\{\, j \mid y_i(j) \le m, y_i(j+1) > m \}$$

搜索策略：深度优先

界：max，m 张邮票可付的连续区间的最大邮资

连续邮资问题：实例 n=4，m=3



解：X=<1,4,7,8>，最大连续区间为{1, …, 24}

个人作业分配问题

一组线性序的人的集合$P=\{P_1, P_2, \ldots, P_n\}$，其中$P_1<P_2<\ldots<P_n$

一组偏序的作业集合$J=\{J_1, J_2, \ldots, J_n\}$

- 设P_i和P_j被分配到任务$f(P_i)$和$f(P_j)$，如果$f(P_i) \le f(P_j)$，则$P_i \le P_j$。
- 设C_{ij}是把作业J_j分配给P_i的费用。
- 我们要找一个使费用最小的可行的作业安排。
- 即求矩阵X，其中：

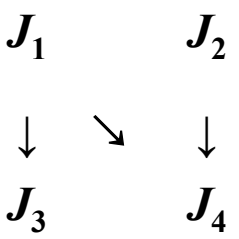
  $X_{ij} = 1$ 表示P_i被分配到任务J_j

  $X_{ij} = 0$ 否则

  使得$\sum_{i,j} C_{ij}X_{ij}$最小。

个人作业分配问题：拓扑排序

- 例：一组偏序的任务



- 拓扑排序后，会产生下列中的一组序列：

J_1, J_2, J_3, J_4

J_1, J_2, J_4, J_3

J_1, J_3, J_2, J_4

J_2, J_1, J_3, J_4

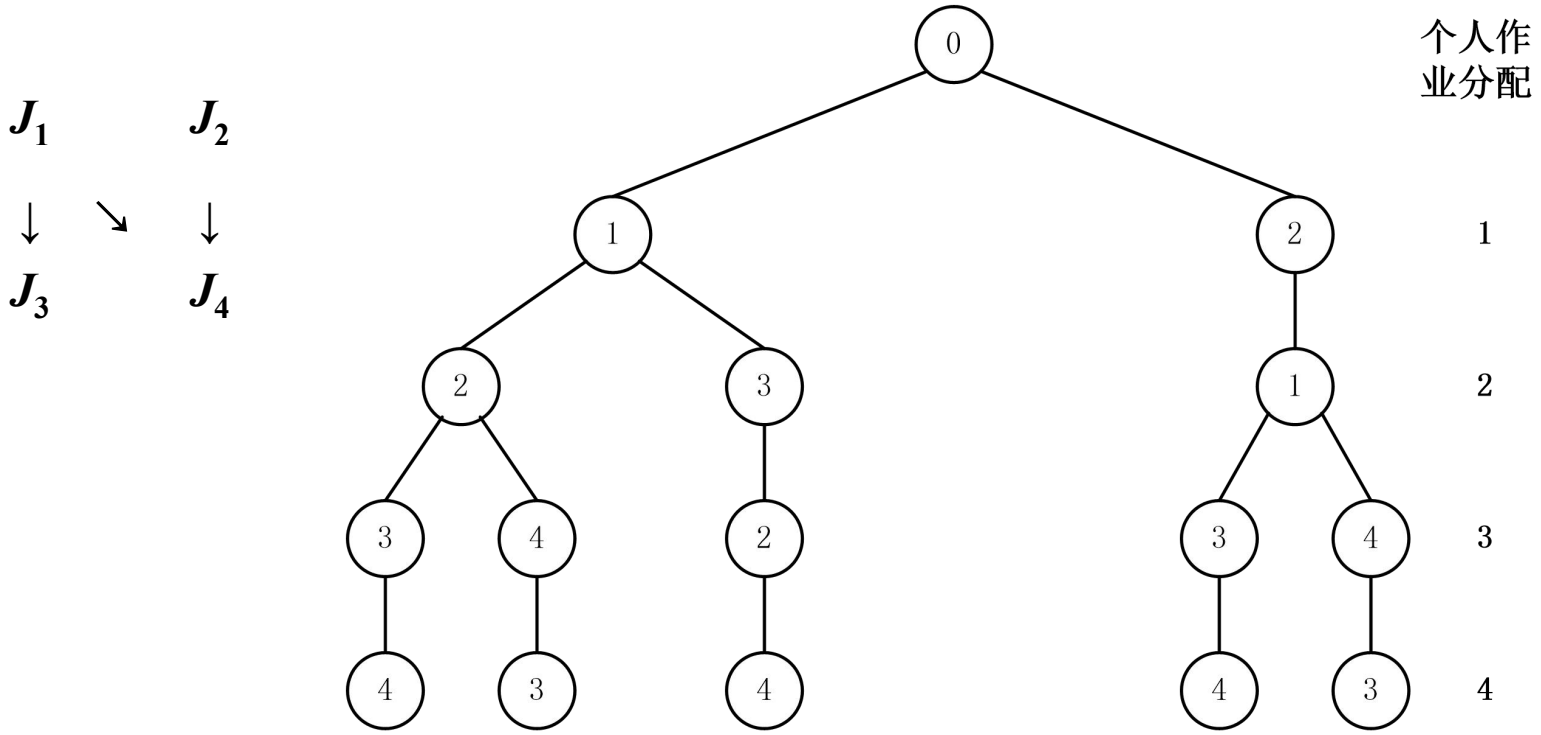
J_2, J_1, J_4, J_3

- 一种可行的任务安排：

$P_1 \rightarrow J_1$, $P_2 \rightarrow J_2$, $P_3 \rightarrow J_3$, $P_4 \rightarrow J_4$

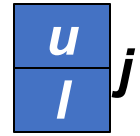
个人作业分配问题：解空间树

- 所有可能的解都可通过一个遍历解空间树给出



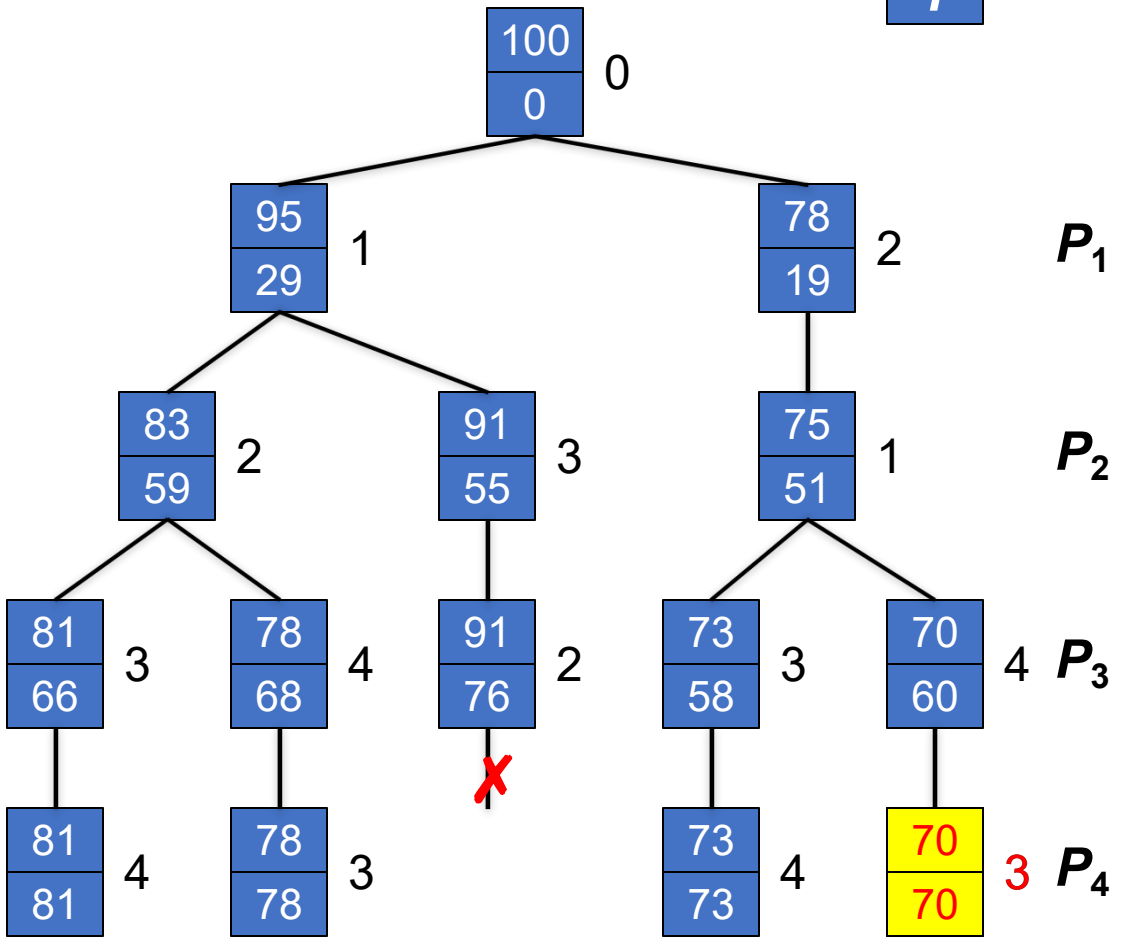
个人作业分配问题：费用矩阵

采用最佳优先搜索方式：Best-First-Search



费用矩阵

作业 / 人	1	2	3	4
1	29	19	17	12
2	32	30	26	28
3	3	21	7	9
4	18	13	10	15



仅一个分支被剪除

个人作业分配问题：费用矩阵化简

费用矩阵

作业 / 人	1	2	3	4
1	29	19	17	12
2	32	30	26	28
3	3	21	7	9
4	18	13	10	15

化简的费用矩阵

作业 / 人	1	2	3	4	
1	17	4	5	0	(-12)
2	6	1	0	2	(-26)
3	0	15	4	6	(-3)
4	8	0	0	5	(-10)
		(-3)			

个人作业分配问题：费用矩阵化简

- 可以获得一个化简的费用矩阵：

  从每行/每列减去相应的常数，使得每行/每列都至少包含一个0项。

- 总的减去的费用为：

  12+26+3+10+3 = 54

- 这个数值是所有解的下界。

个人作业分配问题：简化费用矩阵上用分支限界

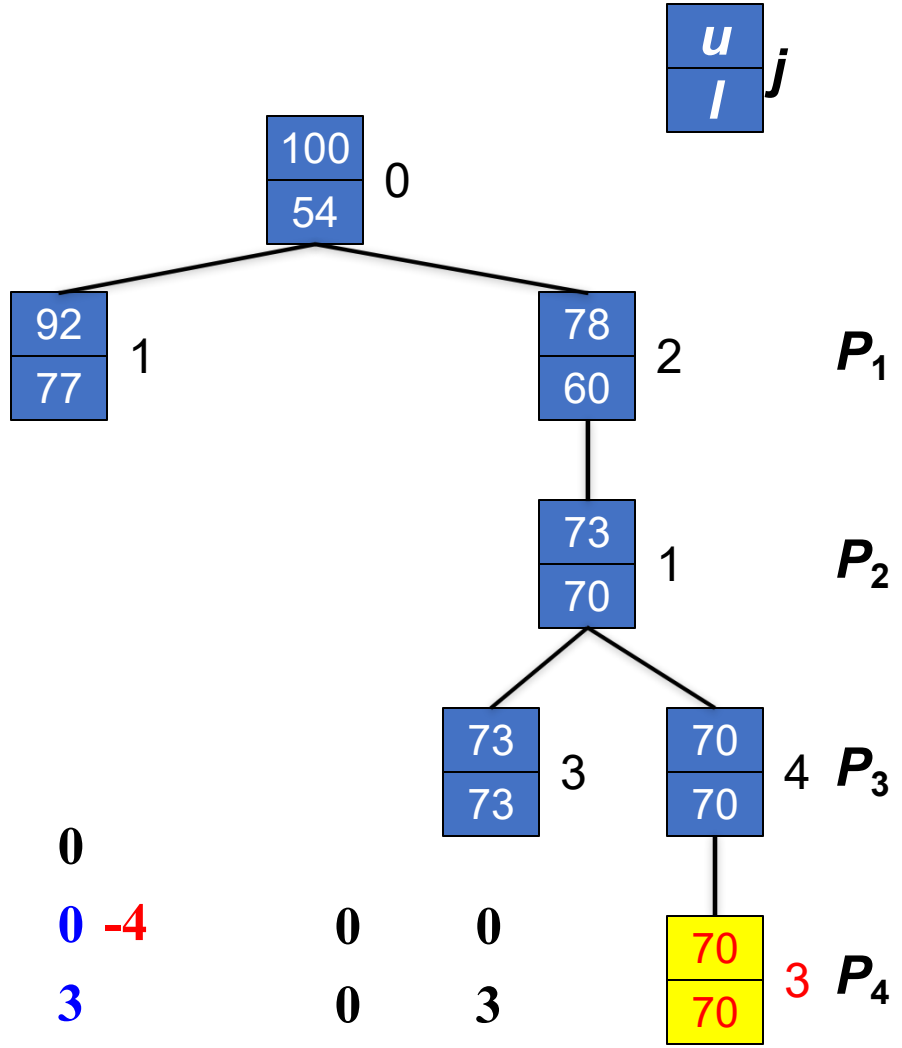
人 \ 任务	1	2	3	4
1	17	4	5	0
2	6	1	0	2
3	0	15	4	6
4	8	0	0	5

1	0	0	
11	0	0	-4
0	0	3	
		-2	

6	0	0	
0	4	4	
8	0	3	
		-2	

6	0	0	
0	0	0	-4
8	0	3	

0	0
0	3



分支限界法基本思想

- 对问题的解有费用函数：$f(x)$
- 对问题的解空间树，分支限界法以广度优先（或以最小费用优先）的方式搜索。
- 分支：每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。
- 首先舍弃所有导致不可行解的儿子结点。对最其他儿子结点，计算结点费用函数$f(x)$的上界$u(x)$和下界$l(x)$，加入当前扩展结点表。
- 限界：扩展结点表中，如果一个结点x的下界$l(x)$大于或等于其它某个结点y的上界$u(y)$，则基于结点x不可能扩展出最优解，可舍弃之。
- 此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。直到当前扩展结点表只剩下一个结点或当前扩展结点集合的上界与下界相等时（则集合中任意结点扩展出的任意解均为最优解）。

分支限界法的最佳实现方式

- 优先队列式分支限界法 - LCBB
  - 按照规定的结点费用最小（LC: ***l*(*x*)最小**）优先的原则选取下一个结点为扩展结点（常采用优先队列实现）。
  - 当扩展结点为叶结点时（$f(x)=u(x)=l(x)$），队列中所有结点的下界$l(y)\geq l(x)=f(x)$，所以此结点即为最优解。
    也就是说：第一个被扩展的叶结点即最优解。
  - 对于优先队列中的任意结点x，如果存在结点y使得$l(y)\geq u(x)$，则y不再可能被选中为扩展结点。相当于y已经被从扩展结点表中删除。

分支限界策略

- 两种机制:
  - 产生分支的机制
  - 产生界的机制
    - 目标：使更多的分支通过界的限制终止
- 平均情形下都很高效，很多分支可以较早剪去
- 虽然通常高效，但最坏情形下搜索空间非常大
- 对NPC问题
  - 很多NPC问题可以用分支限界方法高效的求解（平均情形下），但最坏情形下的复杂性仍然是指数级的

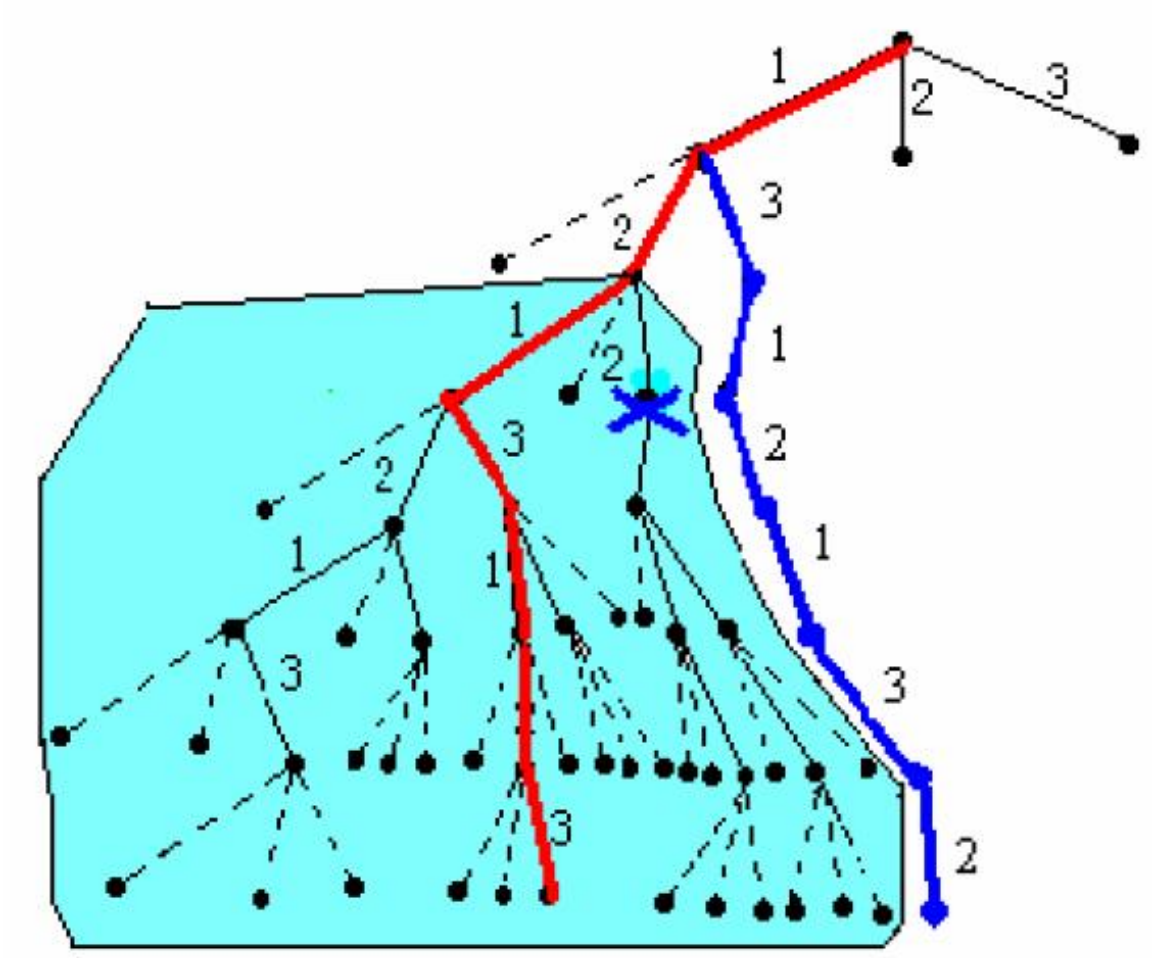
分支限界法与回溯法对比

- 求解目标不同：
  - 回溯法：找一个可行解、找所有可行解、找最优解
  - 分支限界法：找最优解、找一个可行解
- 搜索方式不同：
  - 回溯法：深度优先（剪枝：约束条件、代价函数）
  - 分支限界法：LC, BFS, DFS 均为可选策略（剪枝：上/下界、约束条件）
- 主要特点：
  - 回溯法：空间效率高
  - 分支限界法：往往更“快”

回溯：图的*m*着色

- 给定无向连通图*G*和*m*种颜色，给图的顶点着色，每个顶点一种颜色，且每条边的两个顶点不同色。给出所有可能的着色方案（如果不存在，则回答“No”）
- 搜索空间
  - 为*m*叉完全树。将颜色编号为1, 2, … , *m*。
- 结点表示 $<x_1, x_2, \dots, x_k>$
  - 顶点1着色x_1,顶点2着色x_2, … ， 顶点*k*着色x_k
- 约束条件
  - 该顶点邻接表中已着色的顶点没有同色的
- 代价函数：没有（不是最优化问题）
- 时间：$O(nm^n)$

回溯算法的图示



红色路径的部分解：

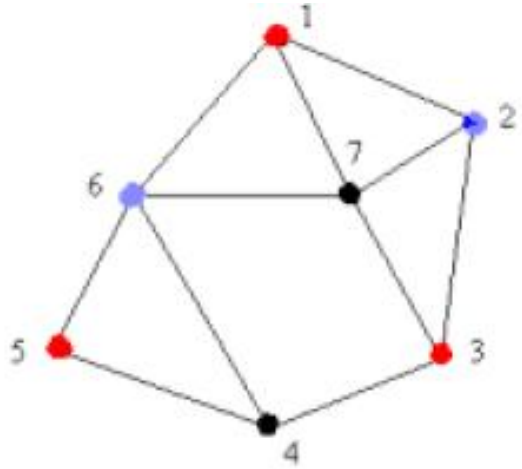
<1红>
<1红,2蓝>
<1红,2蓝,3红>
<1红,2蓝,3红,4黑>
<1红,2蓝,3红,4黑,5红>
<1红,2蓝,3红,4黑,5红,6蓝>
<1红,2蓝,3红,4黑,5红,6蓝,7黑>

观察：蓝色路径是对称的子树

<1红,2蓝,...>
<1红,2黑,...>

观察：若不幸在3节点走错分支

<1红,2蓝,3黑,...> ×

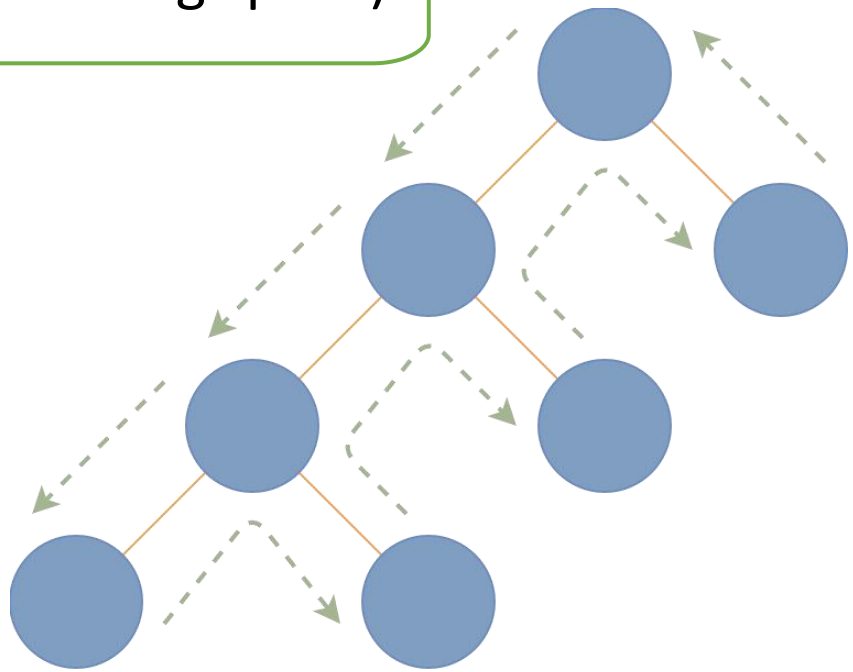


提高效率的途径

- 根据对称性
  - 只需搜索1/3的解空间即可。当1和2确定,即<1,2>以后，只有1个解，因此在<1,3>为根的子树中也只有1个解。由于3个子树的对称性，总共有6个解.
- 提前回溯
  - 进一步分析，在取定<1,2>以后，不可以扩张成<1,2,3>，因为可以检查是否有和1,2,3都相邻的顶点。
  - 如果存在（例子中的点7），则没有解。
  - 所以可以从打叉的结点回溯，而不必搜索它的子树。

SAT 问题的经典回溯算法

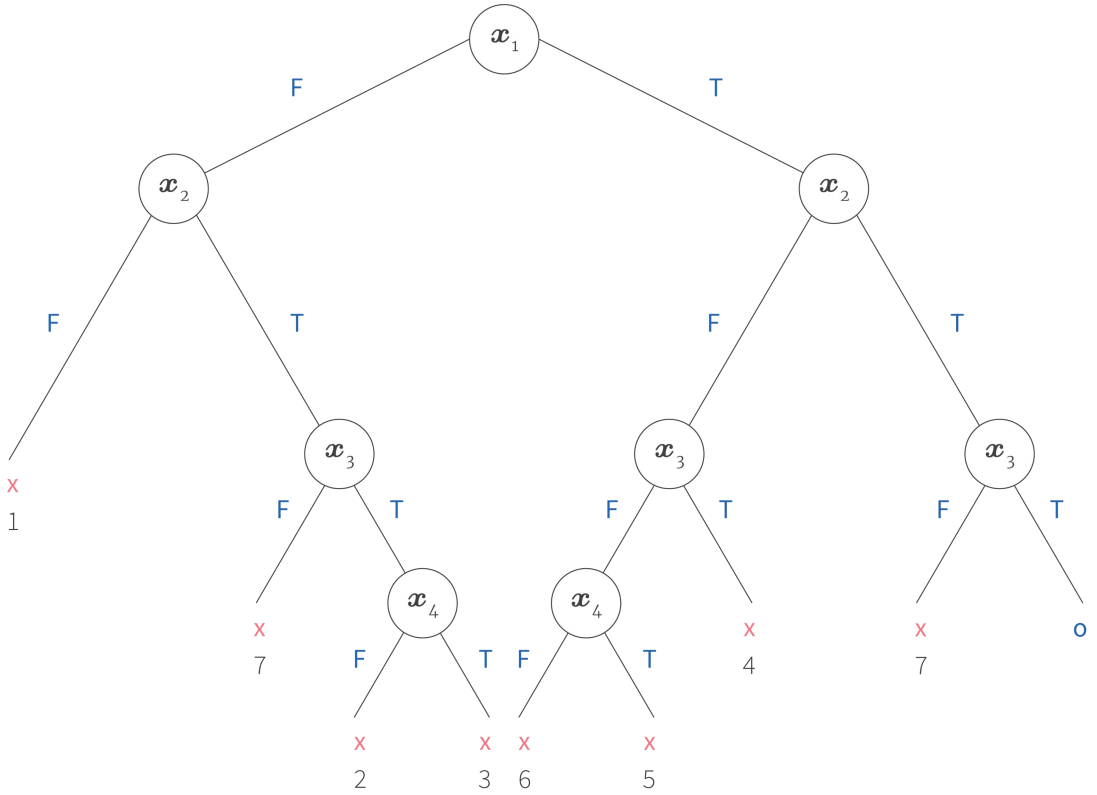
- Davis, Putnam, Logemann and Loveland (DPLL)
- A backtrack search procedure (on 2^n searching space)
- Divide-and-Conquer principle



SAT 问题：直接回溯

- 实例

$$
\begin{aligned}
&1 : (x_1 \vee x_2)\\
&2 : (x_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4)\\
&3 : (x_1 \vee \overline{x}_3 \vee \overline{x}_4)\\
&4 : (\overline{x}_1 \vee x_2 \vee \overline{x}_3)\\
&5 : (\overline{x}_1 \vee x_2 \vee \overline{x}_4)\\
&6 : (\overline{x}_1 \vee x_3 \vee x_4)\\
&7 : (\overline{x}_2 \vee x_3)
\end{aligned}
$$

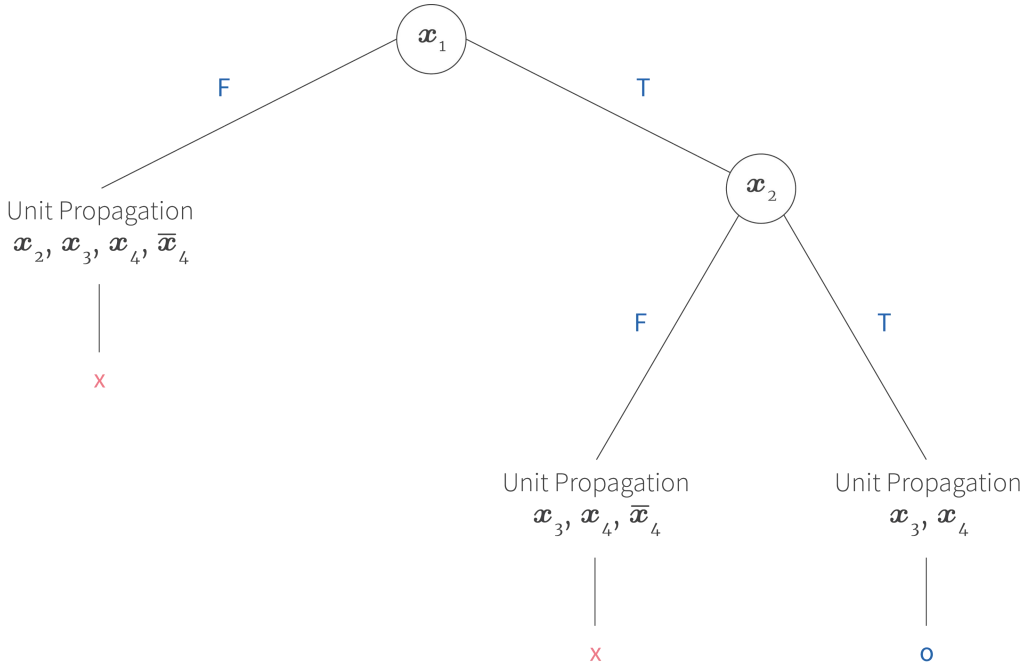


https://www.borealisai.com/en/blog/tutorial-10-sat-solvers-ii-algorithms/

SAT 问题： DPLL 回溯

- 实例

$$
\begin{aligned}
&1 : (x_1 \vee x_2)\\
&2 : (x_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4)\\
&3 : (x_1 \vee \overline{x}_3 \vee \overline{x}_4)\\
&4 : (\overline{x}_1 \vee x_2 \vee \overline{x}_3)\\
&5 : (\overline{x}_1 \vee x_2 \vee \overline{x}_4)\\
&6 : (\overline{x}_1 \vee x_3 \vee x_4)\\
&7 : (\overline{x}_2 \vee x_3)
\end{aligned}
$$



https://www.borealisai.com/en/blog/tutorial-10-sat-solvers-ii-algorithms/

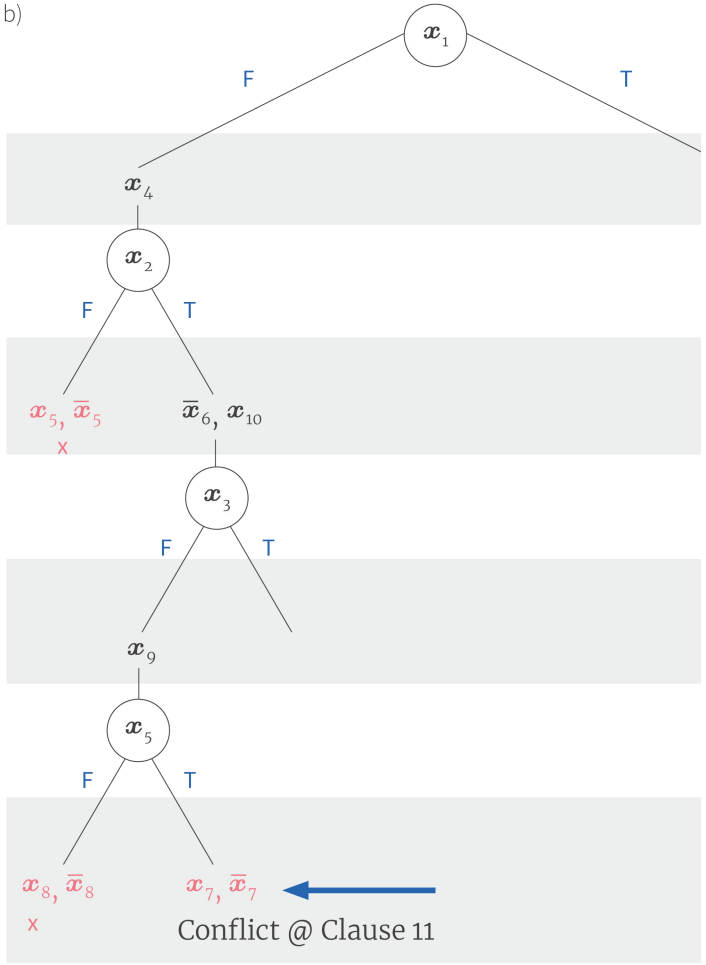
Modern (Sequential) SAT Solvers

- Based on DPLL search procedure combined with
  I. Restart policies
  II. Variable selection heuristics
  III. Highly optimized BCP procedure (~80% runtime)
  IV. Clause learning (CDCL)
- Can now handle millions of variables or more

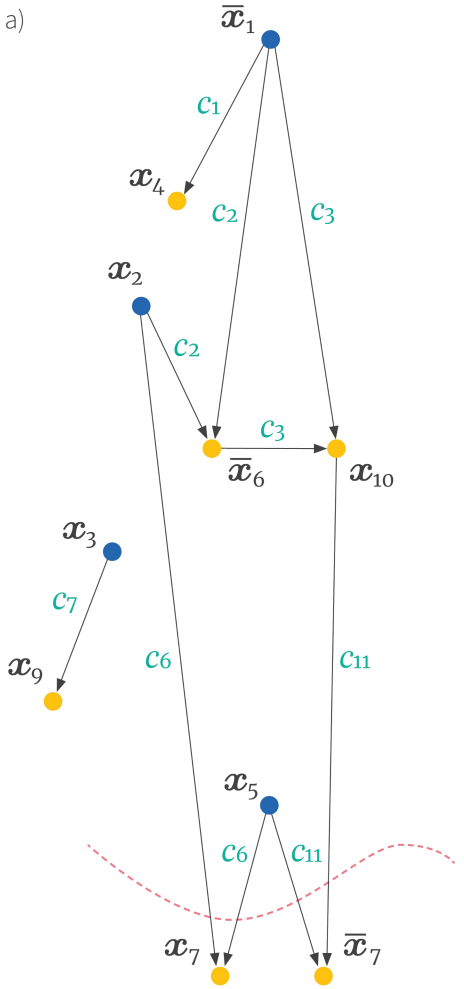
SAT 问题： Conflict-Driven Clause Learning (CDCL)

a) $1: x_1 \vee x_4$
$2: x_1 \vee \bar{x}_2 \vee \bar{x}_6$
$3: x_1 \vee x_6 \vee x_{10}$
$4: x_2 \vee x_5$
$5: x_2 \vee \bar{x}_5$
$6: \bar{x}_2 \vee \bar{x}_5 \vee x_7$
$7: x_3 \vee x_9$
$8: x_5 \vee x_6 \vee x_8$
$9: x_5 \vee x_6 \vee \bar{x}_8$
$10: x_5 \vee x_8 \vee \bar{x}_{10}$
$11: \bar{x}_5 \vee \bar{x}_7 \vee \bar{x}_{10}$

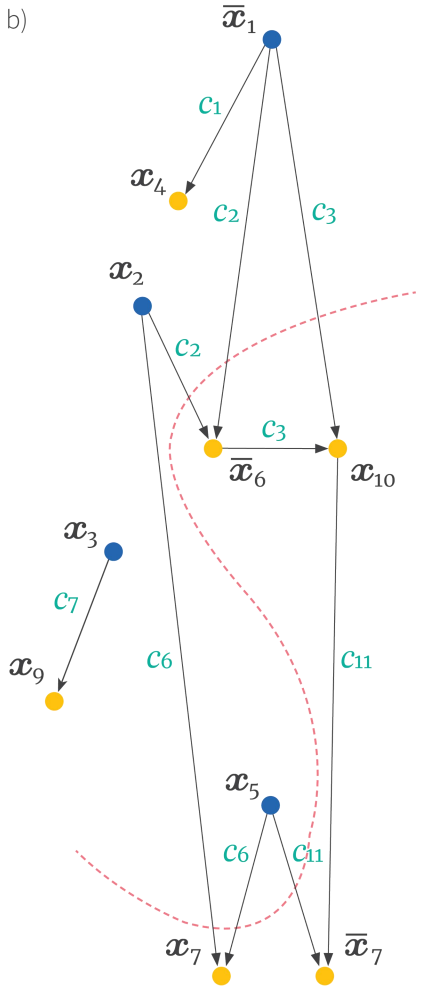
b)



a)

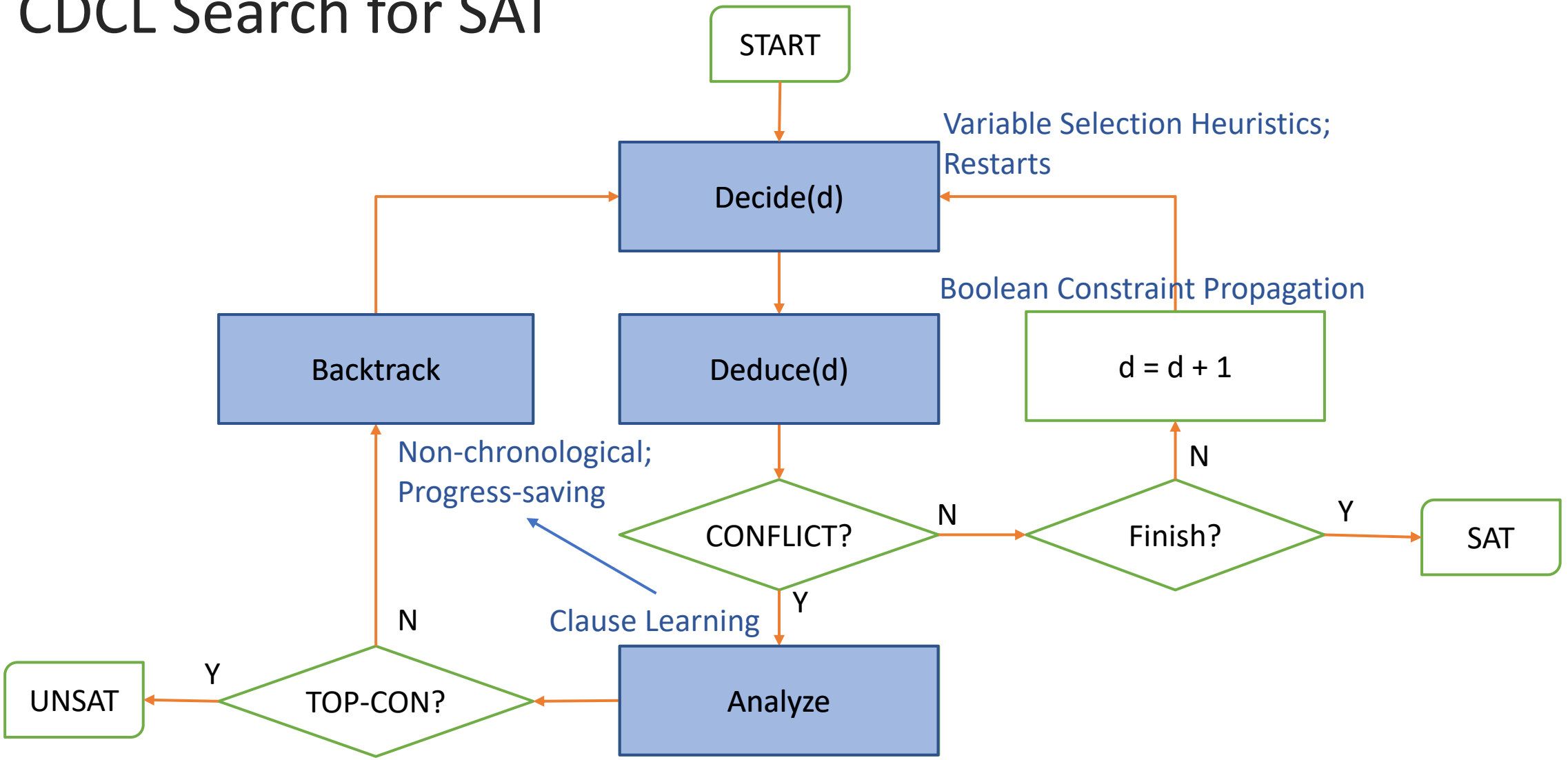


b)

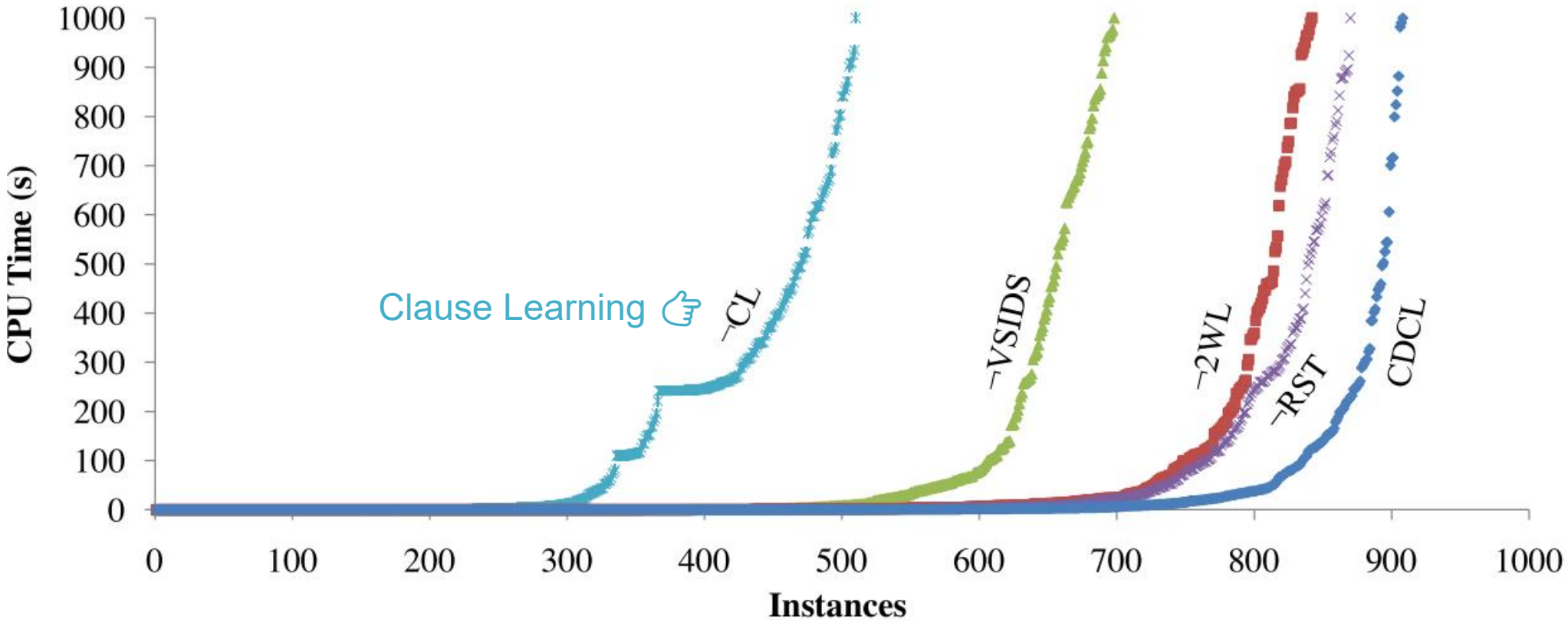


https://www.borealisai.com/en/blog/tutorial-10-sat-solvers-ii-algorithms/

CDCL Search for SAT



Do the features really help?



H. Katebi, K. A. Sakallah, J. P. Marques-Silva. Empirical Study of the Anatomy of Modern Sat Solvers. LNCS 2011.

SAT & CDCL

- 251行代码，即可实现带 CDCL 算法的现代 SAT 求解器

```
206   int i; for (i = 1; i <= n; i++) {                       // Initialize the main datastructures:
207     S->prev [i] = i - 1; S->next[i-1] = i;                // the double-linked list for variable-move-to-front,
208     S->model[i] = S->false[-i] = S->false[i] = 0;         // the model (phase-saving), the false array,
209     S->first[i] = S->first[-i] = END; }                   // and first (watch pointers).
210   S->head = n; }                                          // Initialize the head of the double-linked list
211
212 static void read_until_new_line (FILE * input) {
213   int ch; while ((ch = getc (input)) != '\n')
214     if (ch == EOF) { printf ("parse error: unexpected EOF"); exit (1); } }
215
216 int parse (struct solver* S, char* filename) {                        // Parse the formula and initialize
217   int tmp; FILE* input = fopen (filename, "r");                       // Read the CNF file
218   while ((tmp = getc (input)) == 'c') read_until_new_line (input);
219   ungetc (tmp, input);
220   do { tmp = fscanf (input, " p cnf %i %i \n", &S->nVars, &S->nClauses);  // Find the first non-comment line
221     if (tmp > 0 && tmp != EOF) break; tmp = fscanf (input, "%*s\n"); }    // In case a commment line was found
222   while (tmp != 2 && tmp != EOF);                                     // Skip it and read next line
223
224   initCDCL (S, S->nVars, S->nClauses);                    // Allocate the main datastructures
225   int nZeros = S->nClauses, size = 0;                     // Initialize the number of clauses to read
226   while (nZeros > 0) {                                    // While there are clauses in the file
227     int ch = getc (input);
228     if (ch == ' ' || ch == '\n') continue;
229     if (ch == 'c') { read_until_new_line (input); continue; }
230     ungetc (ch, input);
231     int lit = 0; tmp = fscanf (input, " %i ", &lit);      // Read a literal.
232     if (tmp == EOF) {
233       printf ("s parse error: header incorrect\n"); exit (0); }
234     if (!lit) {                                           // If reaching the end of the clause
235       int* clause = addClause (S, S->buffer, size, 1);    // Then add the clause to data_base
236       if (!size || ((size == 1) && S->false[clause[0]]))  // Check for empty clause or conflicting unit
237         return UNSAT;                                     // If either is found return UNSAT
238       if ((size == 1) && !S->false[-clause[0]]) {         // Check for a new unit
239         assign (S, clause, 1); }                          // Directly assign new units (forced = 1)
240       size = 0; --nZeros; }                               // Reset buffer
241     else S->buffer[size++] = lit; }                       // Add literal to buffer
242   fclose (input);                                         // Close the formula file
243   return SAT; }                                           // Return that no conflict was observed
244
245 int main (int argc, char** argv) {                                  // The main procedure for a STANDALONE solver
246   struct solver S;                                                  // Create the solver datastructure
247   if      (argc  <= 1) {printf("no input file provided\n"); exit (0);} // Expect a single argument
248   else if (parse (&S, argv[1]) == UNSAT) printf("s UNSATISFIABLE\n");  // Parse the DIMACS file in argv[1]
249   else if (solve (&S)          == UNSAT) printf("s UNSATISFIABLE\n");  // Solve without limit (number of conflicts)
250   else                                   printf("s SATISFIABLE\n")  ;  // And print whether the formula has a solution
251   printf ("c statistics of %s: mem: %i conflicts: %i max_lemmas: %i\n", argv[1], S.mem_used, S.nConflicts, S.maxLemmas); }
```

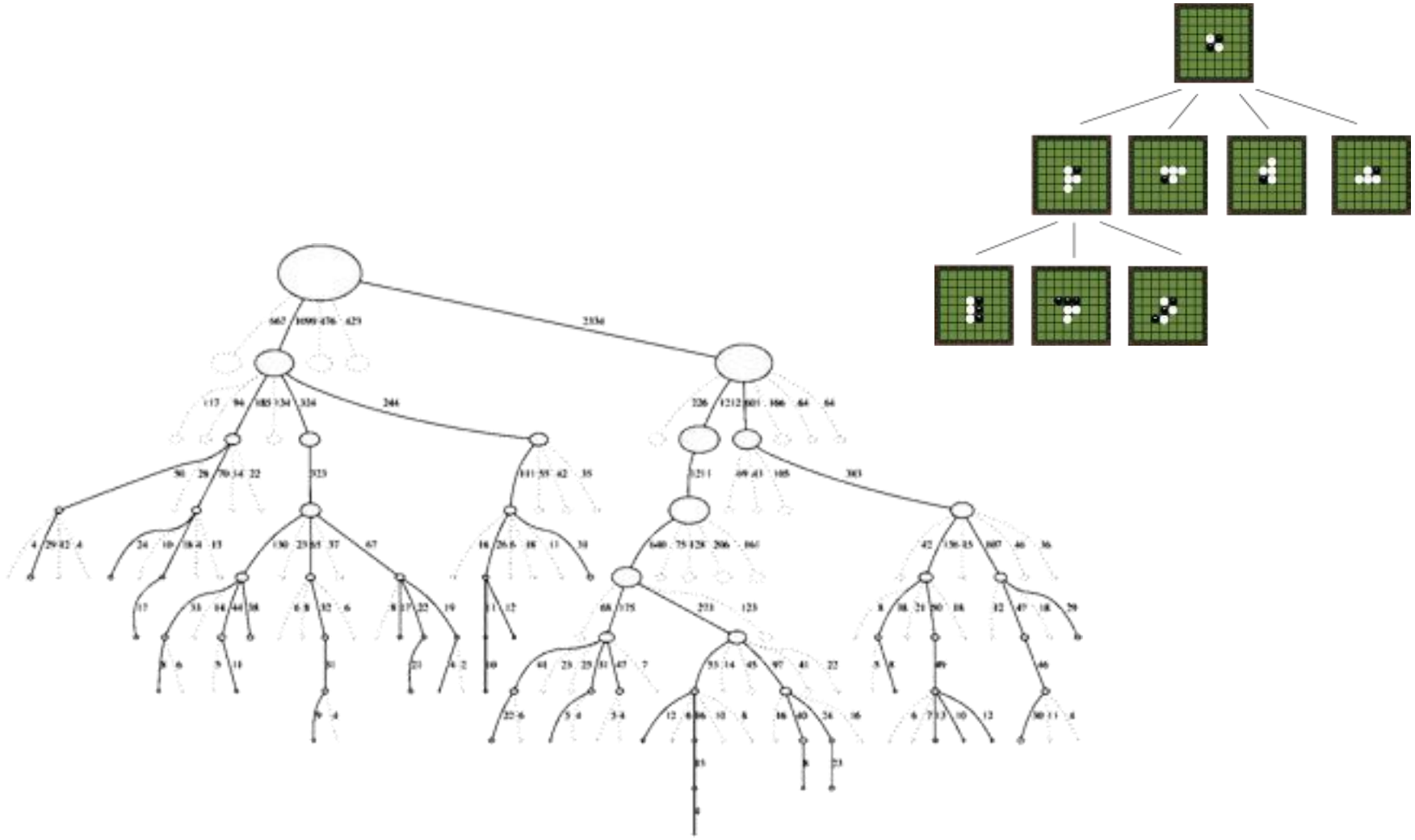
https://github.com/marijnheule/microsat/blob/master/microsat.c

改进途径

- 利用搜索树的对称性剪裁子树
- 根据树分支设计优先策略
  - 结点少的分支优先
  - 解多的分支优先

- 分解为子问题:
  - 问题求解时间$f(n)=c2^n$，子问题组合时间$T=O(n)$
  - 如果分解为k个子问题，每个子问题大小为n/k
  - 则求解时间为$kc2^{n/k}+T$

A Well-known Example of Discrete Search



小结

- **适应于求解组合搜索问题**（含组合优化问题）
- **求解条件：**满足多米诺性质
- **解的表示：**解向量，求解是不断扩充解向量的过程
- **回溯条件：**
  - 搜索问题——约束条件
  - 优化问题——约束条件 + 代价函数
- **算法复杂性：**
  - 遍历搜索树的时间，最坏情况为指数
  - 空间代价小
- **平均时间复杂性的估计：**Monte Carlo方法
- **降低时间复杂性的主要途径：**
  - 利用对称性裁减子树
  - 划分成子问题
- **分支策略**（深度优先、宽度优先、宽深结合、优先函数）