两级逻辑优化

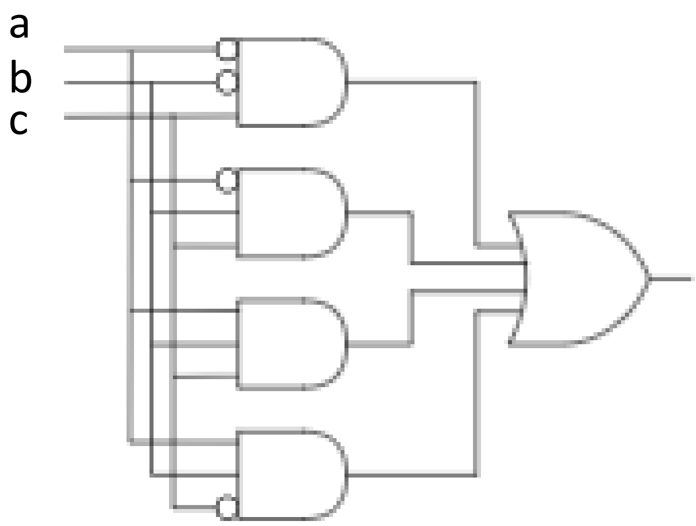
Two-level Logic Optimization

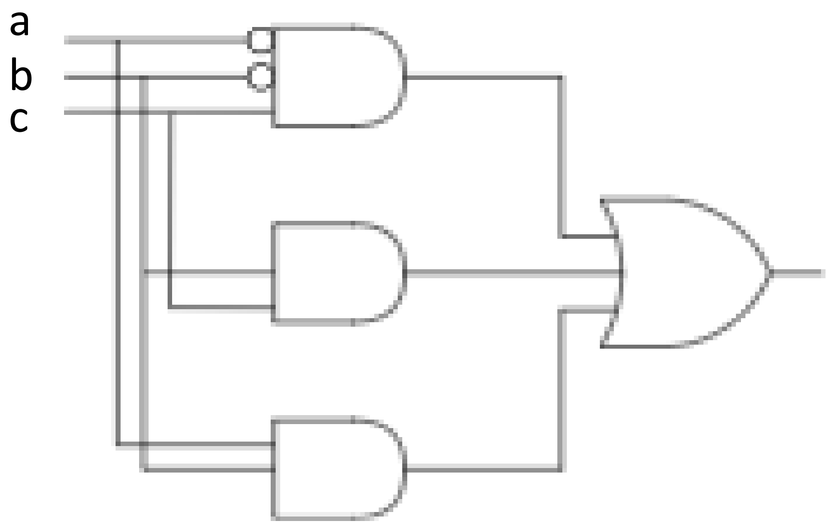
2024-3

主讲人：孙静翎

两级逻辑优化

Two-level logic optimization





x = a’ b’ c + a’ bc + abc +abc’

x = a’ b’ c + *bc + ab*

例子

假设公司里有A、B、C、D四个员工，公司总共需要做5件事情，有哪些裁员方案?

- A：125
- B：123
- C：245
- D：124
- E：24

定义

- 最小覆盖（Minimum cover）：
  - 具有最少蕴含项（implications）的函数覆盖
  - 全局最优
- 非冗余覆盖（Minimal cover or irredundant cover）：
  - 不完全是另一个函数覆盖的超集
  - 不能删除任何蕴含项
  - 局部最优
- 单个蕴含项下的非冗余覆盖（Minimal w.r.t. single-implicant containment）：
  - 没有蕴含项包含另一个蕴含项
  - 弱局部最优

例子

假设公司里有A、B、C、D四个员工，公司总共需要做5件事情，有哪些裁员方案?

- A：125
- B：123
- C：245
- D：124
- E：24

最小覆盖：BC

非冗余覆盖：BC，ABD，ABE

单个蕴含项下的非冗余覆盖：BC，ABC，ABD，ABE，BCD，BCE，ABCD

两级逻辑优化

Two-level logic minimization

- 精确逻辑最小化：
  - 目标是计算出一个最小覆盖。对于大型函数是困难的
  - Quine-McCluskey方法（奎因-麦克拉斯基算法）
- 启发式逻辑最小化：
  - 致力于在短时间内计算近似的最小覆盖，这通常足够快速但可能不是最优解。
  - MINI, PRESTO, ESPRESSO

精确的逻辑最小化

Exact logic minimization

- Quine定理：
    - 最小覆盖一定是质覆盖
- 因此：
    - 可以在质蕴含项中搜索最小覆盖
- Quine-McCluskey（奎因-麦克拉斯基算法）方法
    - 计算质蕴含项
    - 确定最小覆盖

计算质蕴含项

1. 确认一次蕴含项，例如：

$$F=AB'CD+A'B'CD'+A'BCD'+ABCD'+AB'CD'+A'BC'D$$

1011

0010

0110

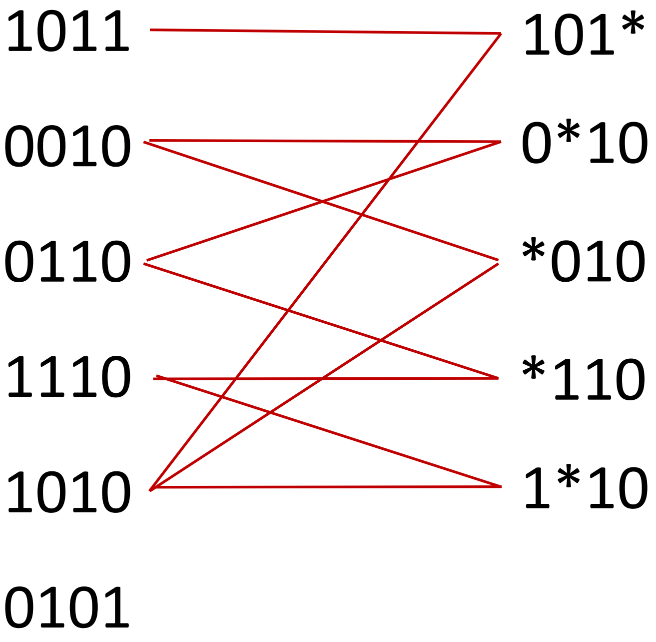
1110

1010

0101

计算质蕴含项

1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量



计算质蕴含项

1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量
3. 删除所有被用于合并的蕴含项和重复的蕴含项

1011	101*
0010	0*10
0110	*010
1110	*110
1010	1*10
0101	

重复步骤2和步骤3直到没有剩余的蕴含项可以被合并

计算质蕴含项

1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量
3. 删除所有被用于合并的蕴含项和重复的蕴含项

1011	101*	
0010	0*10	**10
0110	*010	**10
1110	*110	
1010	1*10	
0101		

重复步骤2和步骤3直到没有剩余的蕴含项可以被合并

计算质蕴含项

1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量
3. 删除所有被用于合并的蕴含项和重复的蕴含项

1011	101*	
0010	0*10	**10
0110	*010	**10
1110	*110	
1010	1*10	
0101		

重复步骤2和步骤3直到没有剩余的蕴含项可以被合并

计算质蕴含项

1. 确认一次蕴含项
2. 对每两个蕴含项进行对比，如果只有一个变量不同则合并产生新的蕴含项，新的蕴含项保留两个蕴含项相同的变量，并用-替换他们间不同的变量
3. 删除所有被用于合并的蕴含项和重复的蕴含项

1011	101*	
0010	0*10	**10
0110	*010	**10
1110	*110	
1010	1*10	
0101		

重复步骤2和步骤3直到没有剩余的蕴含项可以被合并

最终得到的质蕴含项：

0101

101*

**10

一个小优化

0010

0110

1010

0101

1110

1011

最低覆盖率的早期方法

Minimum cover early methods

- 简化表格
  - 迭代地识别必要元素，将它们保存在覆盖中。
  - 移除已覆盖的最小项。
- Petrick 方法
  - 以积之和（POS）形式写覆盖子句
  - 将积之和形式展开为和之积（SOP）的形式
  - 选择最小的立方项
  - 注意：展开子句的成本是指数级的

质蕴含项表

Prime implicant table

- 质蕴含项表是一个二进制矩阵表格，表格的：
  - 列 表示所有的质蕴含项
  - 行 表示所有需要被覆盖的最小项

- 在表格中用 ✓（或 1）标记哪些质蕴含项可以覆盖哪些最小项。

例子

Example

- $f = a'b'c' + a'b'c + ab'c + abc + abc'$
- 蕴含项表:

	abc	f
α	00*	1
β	*01	1
γ	1*1	1
δ	11*	1

- 质蕴含项表:

	α(a'b')	β(b'c)	γ(ac)	δ(ab)
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

通过简化表格确定最小覆盖

1. 对所有只包含一项"1"的行（图中的红色标记），对其项“1”所在的列画竖线，并对竖线所经过的项“1”画横线

	α(a’b’)	β(b’c)	γ(ac)	δ(ab)
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

通过简化表格确定最小覆盖

2. 找出为包含最大数量未被划线的项“1”的列，若有多列，则必有多种化简结果

对其划竖线，并对当前项1及竖线所经过的项“1”画横线

	α(a’b’)	β(b’c)	γ(ac)	δ(ab)
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

通过简化表格确定最小覆盖

3. 若经过划线后还存在未划线项，继续步骤2直到无未划线项为止，最后的化简结果（最小覆盖）即为所有划竖线的列对应的项的和

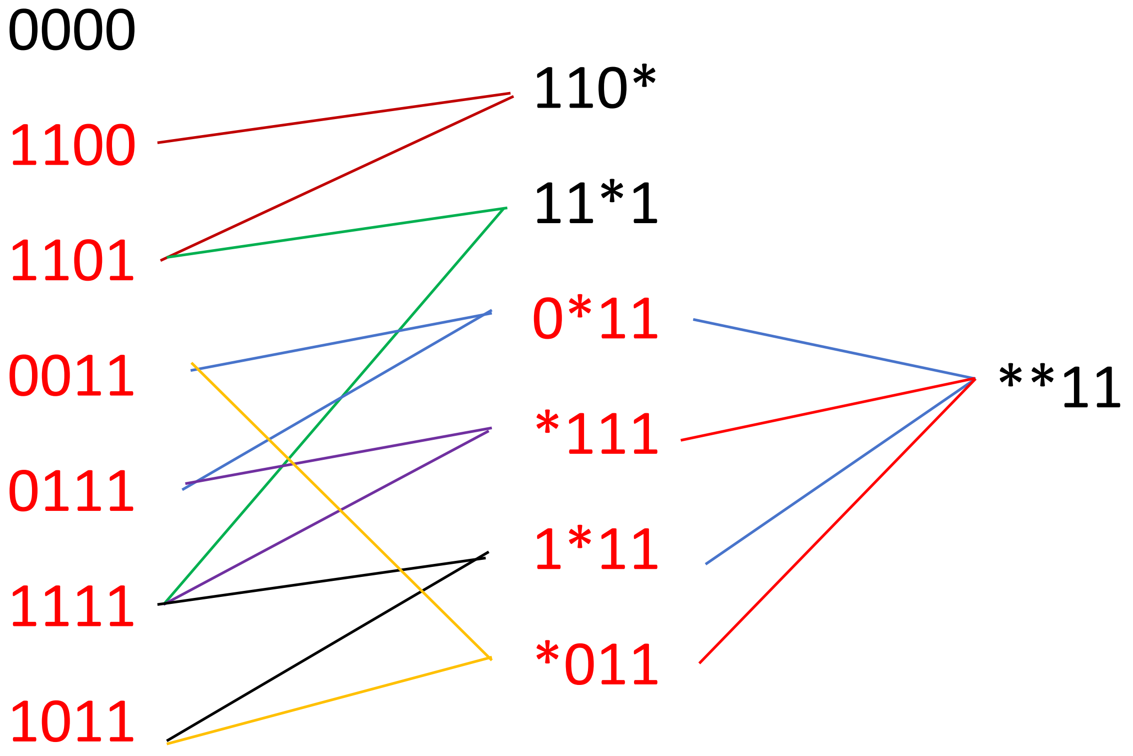
	α(a’b’)	β(b’c)	γ(ac)	δ(ab)
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

$f = a'b' + b'c + ab$

随堂作业

请列出以下布尔函数的质蕴含项表：

F=A’B’C’D’+ABC’D’+ABC’D+A’B’CD+A’BCD+ABCD+AB’CD



	A ‘B’ C ‘D’	ABC’	ABD	CD
0000	1	0	0	0
1100	0	1	0	0
1101	0	1	1	0
0011	0	0	0	1
0111	0	0	0	1
1111	0	0	1	1
1011	0	0	0	1

随堂作业

	A’B’C’D’	ABC’	ABD	CD
0000	1	0	0	0
1100	0	1	0	0
1101	0	1	1	0
0011	0	0	0	1
0111	0	0	0	1
1111	0	0	1	1
1011	0	0	0	1

F=A’B’C’D’+ABC’+CD

正式开始本周的内容

Quine-McCluskey方法-实现

```
# 示例输入
minterms = ['0000', '1100', '1101', '0011', '0111', '1111', '1011']

# 第一步：计算质蕴含项
prime_implicants = get_prime_implicants(minterms)

# 第二步：确定最小覆盖
essentials = get_essential_prime_implicants(prime_implicants, minterms)
print(“\n最小覆盖： ")
for term in essentials:
    print(term)
```

计算质蕴含项-实现

```
def get_prime_implicants(minterms):
    groups = group_by_ones(minterms)  # 按照1的个数对蕴含项进行分组
    prime_implicants = set()
    # 不断尝试将当前分组进行合并，直到无法再合并
    while groups:
            # 合并相邻组的项，返回新产生的蕴含项和已被用于合并的蕴含项
            new_groups, marked = combine_groups(groups)
            # 收集上一轮合并中未被合并的项作为质蕴含项
            prime_implicants.update(collect_unmarked(groups, marked))
            # 对要用于下一轮合并的蕴含项去重
            groups = {k: remove_duplicates(v) for k, v in new_groups.items()}
    # 返回所有质蕴含项
    return list(prime_implicants)
```

按照1的个数对蕴含项进行分组-实现

```python
def group_by_ones(minterms):
    groups = {}
    for m in minterms:
        ones = count_ones(m)
        groups.setdefault(ones, []).append(m)
    return groups
```

```
groups = {0: ['0000'], 2: ['1100', '0011'], 3: ['1101', '0111', '1011'], 4: ['1111']}
```

计算质蕴含项-实现

```python
def get_prime_implicants(minterms):
    groups = group_by_ones(minterms)  # 按照1的个数对蕴含项进行分组
    prime_implicants = set()
    # 不断尝试将当前分组进行合并，直到无法再合并
    while groups:
            # 合并相邻组的项，返回新产生的蕴含项和已被用于合并的蕴含项
            new_groups, marked = combine_groups(groups)
            # 收集上一轮合并中未被合并的项作为质蕴含项
            prime_implicants.update(collect_unmarked(groups, marked))
            # 对要用于下一轮合并的蕴含项去重
            groups = {k: remove_duplicates(v) for k, v in new_groups.items()}
    # 返回所有质蕴含项
    return list(prime_implicants)
```

合并相邻组的项-实现

```python
def combine_groups(groups):
    new_groups = {}
    marked = set()
    keys = sorted(groups.keys())
    for i in range(len(keys) - 1):
        for a in groups[keys[i]]:
            for b in groups[keys[i + 1]]:
                combined = combine_terms(a, b)
                if combined:
                    marked.add(a)
                    marked.add(b)
                    k = count_ones(combined.replace('-', ''))
                    new_groups.setdefault(k, []).append(combined)
    return new_groups, marked
```

0000

1100

0011

1101

0111

1011

1111

计算质蕴含项-实现

```python
def get_prime_implicants(minterms):
    groups = group_by_ones(minterms)  # 按照1的个数对蕴含项进行分组
    prime_implicants = set()
    # 不断尝试将当前分组进行合并，直到无法再合并
    while groups:
            # 合并相邻组的项，返回新产生的蕴含项和已被用于合并的蕴含项
            new_groups, marked = combine_groups(groups)
            # 收集上一轮合并中未被合并的项作为质蕴含项
            prime_implicants.update(collect_unmarked(groups, marked))
            # 对要用于下一轮合并的蕴含项去重
            groups = {k: remove_duplicates(v) for k, v in new_groups.items()}
    # 返回所有质蕴含项
    return list(prime_implicants)
```

确定最小覆盖-实现

```python
def get_essential_prime_implicants(prime_implicants, minterms):
    # 构建覆盖表：每个最小项被哪些质蕴含项覆盖
    …
    # 找出必要质蕴含项和被必要项覆盖的最小项
    …
    # 找出还未被覆盖的最小项和仍未被选择的质蕴含项
    …
    # 尝试从 candidates 中选择最少的项完全覆盖剩余最小项
    …
```

确定最小覆盖-实现

```
def get_essential_prime_implicants(prime_implicants, minterms):
    # 构建覆盖表：每个最小项被哪些质蕴含项覆盖
    chart = {m: [] for m in minterms}
    for m in minterms:
        for p in prime_implicants:
            if is_covered(p, m):
                chart[m].append(p)
    # 找出必要质蕴含项和被必要项覆盖的最小项
    …
    # 找出还未被覆盖的最小项和仍未被选择的质蕴含项
    …
    # 尝试从 candidates 中选择最少的项完全覆盖剩余最小项
    …
```

{'000': [], '001': [], '101': [], '111': [], '110': []}

{'000': ['00-'], '001': ['00-', '-01'], '101': ['-01', '1-1'], '111': ['11-', '1-1'], '110': ['11-']}

确定最小覆盖-实现

```
def get_essential_prime_implicants(prime_implicants, minterms):
    # 构建覆盖表：每个最小项被哪些质蕴含项覆盖     …
    # 找出必要质蕴含项和被必要项覆盖的最小项
    essential = set() # 存储必要质蕴含项
    covered_minterms = set() # 存储已被覆盖的最小项
    for m, ps in chart.items():
        if len(ps) == 1:
            essential.add(ps[0])
    for m in minterms:
        if any(is_covered(p, m) for p in essential):
            covered_minterms.add(m)
    # 找出还未被覆盖的最小项和仍未被选择的质蕴含项     …
    # 尝试从 candidates 中选择最少的项完全覆盖剩余最小项     …
```

确定最小覆盖-实现

```python
def get_essential_prime_implicants(prime_implicants, minterms):
    # 构建覆盖表：每个最小项被哪些质蕴含项覆盖    …
    # 找出必要质蕴含项和被必要项覆盖的最小项
    essential = set() # 存储必要质蕴含项
    covered_minterms = set() # 存储已被覆盖的最小项
    …
    # 找出还未被覆盖的最小项和仍未被选择的质蕴含项
    remaining_minterms = [m for m in minterms if m not in covered_minterms]
    if not remaining_minterms:
        return list(essential)
    candidates = [p for p in prime_implicants if p not in essential]
    # 尝试从 candidates 中选择最少的项完全覆盖剩余最小项
    …
```

确定最小覆盖-实现

```python
def get_essential_prime_implicants(prime_implicants, minterms):
    # 构建覆盖表：每个最小项被哪些质蕴含项覆盖      …
    # 找出必要质蕴含项和被必要项覆盖的最小项      …
    # 找出还未被覆盖的最小项和仍未被选择的质蕴含项      …
    # 尝试从 candidates 中选择最少的项完全覆盖剩余最小项
    from itertools import combinations
    for r in range(1, len(candidates) + 1):
        for subset in combinations(candidates, r):
            covered = set()
            for m in remaining_minterms:
                if any(is_covered(p, m) for p in subset):
                    covered.add(m)
            if len(covered) == len(remaining_minterms):
                return list(essential.union(subset))
```

比较极端的例子

	0*01	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

需要尝试的蕴含项组合：

0*01

010*

…

0*01，010*

*100，1*00

…

0*01，010*，*100

0*01，010*，*100

…

比较极端的例子

	0*01	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

比较极端的例子

	0*01	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

比较极端的例子

	0*01	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

比较极端的例子

	0*01	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

比较极端的例子

	0*01	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

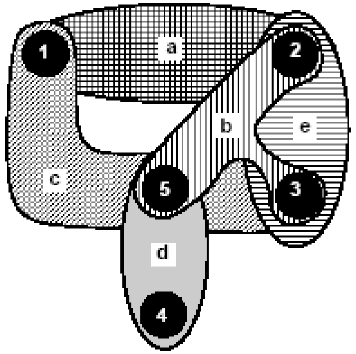
比较极端的例子

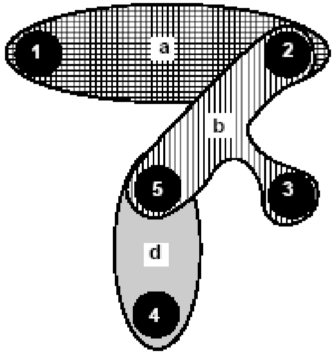
	0*01	010*	*100	1*00	100*	*001
0001	1	0	0	0	0	1
0100	0	1	1	0	0	0
0101	1	1	0	0	0	0
1100	0	0	1	1	0	0
1000	0	0	0	1	1	0
1001	0	0	0	0	1	1

覆盖问题

Covering problem

- 集合覆盖问题
  - 一个集合 S （最小项集）
  - 蕴含项集（implicant set）的集合 C
  - 在C 中选择最少的元素来覆盖 S
- 精确解法
  - Branch and bound algorithm 分枝定界算法
- 多种启发式近似方法



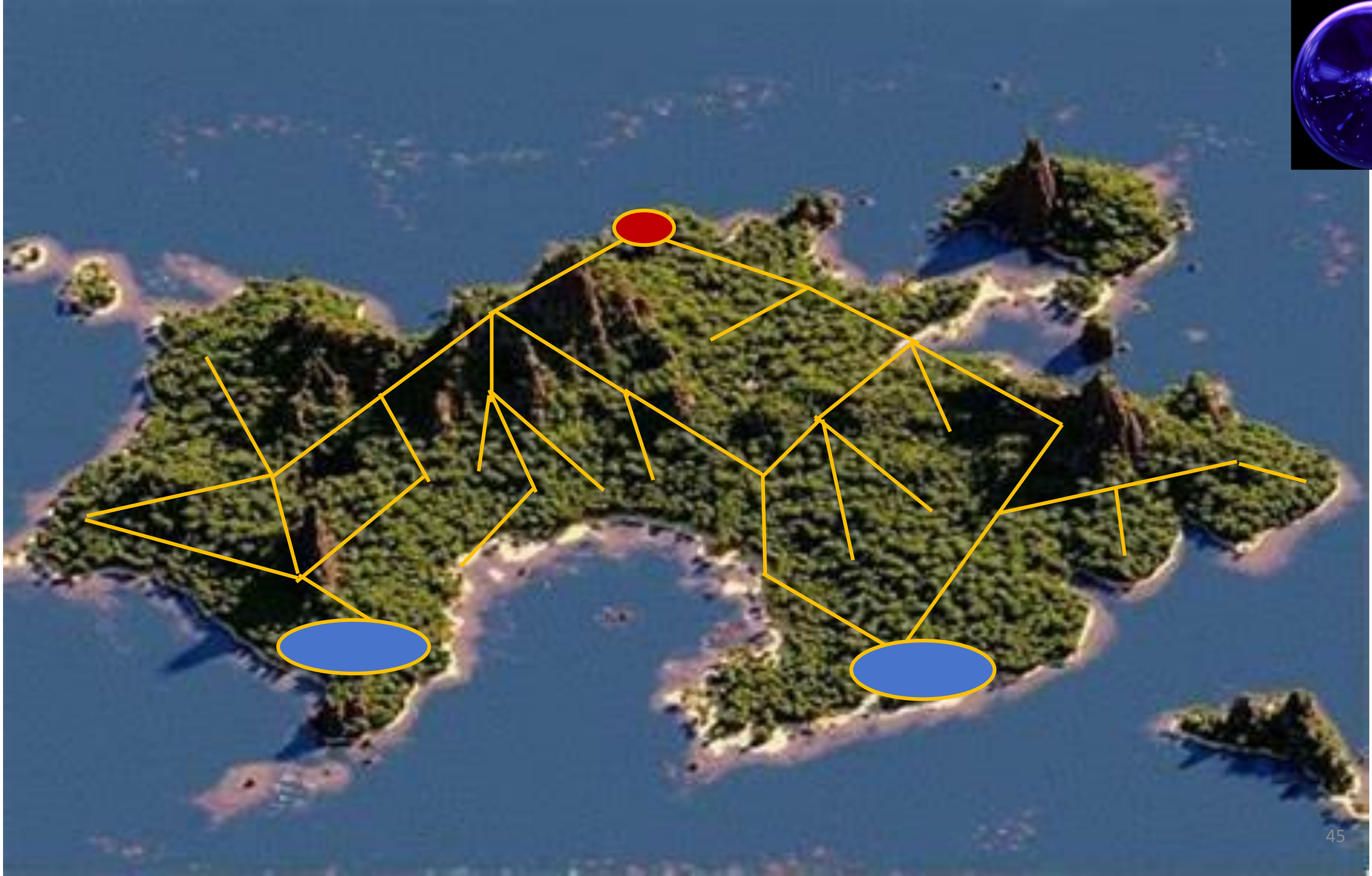


分枝定界算法

Branch and bound algorithm

分枝定界算法

Branch and bound algorithm



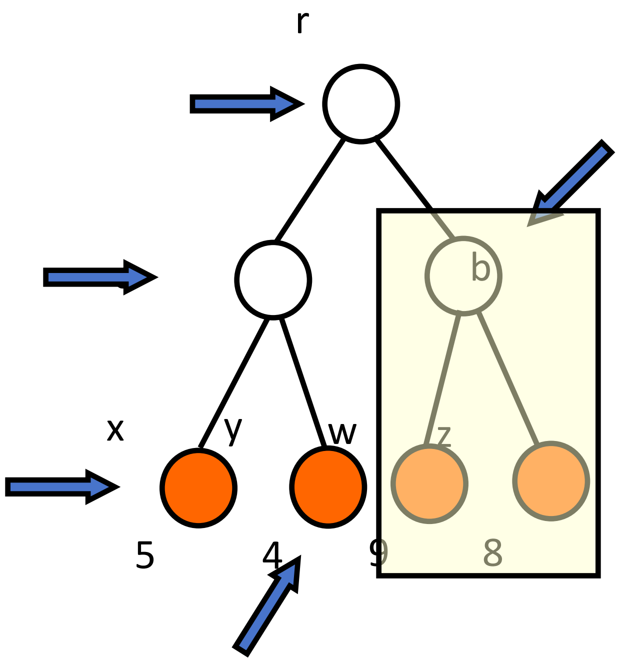
分枝定界算法

Branch and bound algorithm

- 在解决空间树中进行搜索
  - 潜在的指数增长
- 使用界定函数：
  - 如果从一组未来选择中得出的解决方案成本的下限超过了迄今为止看到的最佳解决方案的成本，那么停止该路径的搜索
  - 界定函数必须是可信的并可以快速被验证
- 良好的剪枝可以加快搜索过程

例子

Example



界限 = 6（即要走6天才能找到水）

则放弃这个子树

```
EXACT_COVER(A,x,b) {
   if (current_estimate ≥ |b|) return (b);
   Reduce matrix A and update corresponding x;
   if (A has no rows) return(x);
   select a branching column c;
   x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x~ = EXACT_COVER(Ã,x_c ,b);
    if ( | x~| < |b| )
           b = x~;
    x_c = x;
    Ã = A after deleting c;
    x~ = EXACT_COVER(Ã,x_c ,b);
    if ( | x~| < |b| )
           b = x~;
   return(b);
}
```

```
EXACT_COVER(A, x, b) {
  如果 current_estimate ≥ |b|， 则返回 b;
  对矩阵 A 进行化简并更新对应的 x;
  如果 A 没有任何行，则返回 x;
  选择一个用于分支的列 c;
  xc = x + 1;
  Ã = 删除列 c 以及与其相关的行后的 A;
  x~ = EXACT_COVER(Ã, xc, b);
  如果 |x~| < |b|， 则
    b = x~;
  xc = x;
  Ã = 删除列 c 后的 A;
  x~ = EXACT_COVER(Ã, xc, b);
  如果 |x~| < |b|， 则
    b = x~;
  返回 b;
}
```

	α(a'b')	β(b'c)	γ(ac)	δ(ab)
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

分枝定界算法

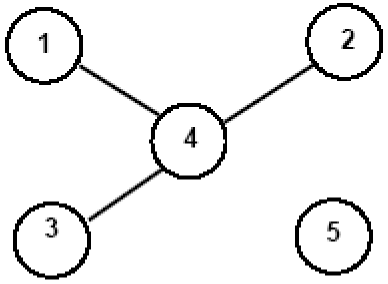
Branch and bound algorithm

- 行的独立关系
  - 行中有1的部分和另一行不重叠，则两行存在独立关系
  - 对每一行都需要独立的蕴含项
- 行的独立图
  - 把每一行看做一个结点
  - 两行如果有独立关系就在独立图中用线连接

例子

- 第四行和1，2，3是独立的

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$



分枝定界算法

Branch and bound algorithm

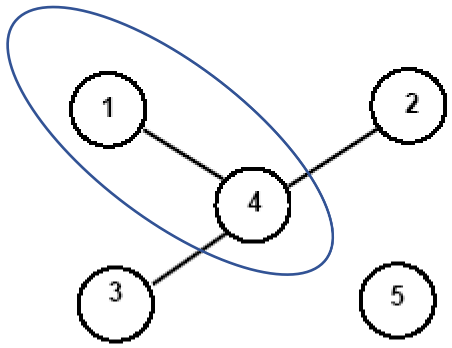
- 行的独立关系
  - 行中有1的部分和另一行不重叠，则两行存在独立关系
  - 对每一行都需要独立的蕴含项
- 行的独立图
  - 把每一行看做一个结点
  - 两行如果有独立关系就在独立图中用线连接
- 找到独立图中尽量大的团
- 团的结点数就是可以接受的近似（下界）

例子

- 第四行和1，2，3是独立的
- 最大的团的节点数是2
- 预测还最少需要的蕴含项是：

  最大团的结点数

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$



```
EXACT_COVER(A, x, b) {
    如果 current_estimate ≥ |b|， 则返回 b;
    对矩阵 A 进行化简并更新对应的 x;
    如果 A 没有任何行，则返回 x;
    选择一个用于分支的列 c;
    xc = x + 1;
    Ã = 删除列 c 以及与其相关的行后的 A;
    x~ = EXACT_COVER(Ã, xc, b);
    如果 |x~| < |b|， 则
        b = x~;
    xc = x;
    Ã = 删除列 c 后的 A;
    x~ = EXACT_COVER(Ã, xc, b);
    如果 |x~| < |b|， 则
        b = x~;
    返回 b;
}
```

缩减矩阵的方法

- 找到基本要素：
  - 如果一行中只有一列为1
    - 选择该列
    - 从表中移除被覆盖的行

	α	β	γ	δ
000	1	0	1	0
001	0	1	1	0
101	1	1	1	0
111	1	0	0	1
110	0	0	0	1

缩减矩阵的方法

	α	β	γ	δ
000	1	0	1	0
001	0	1	1	0
101	1	1	1	0
111	0	0	0	1
110	0	1	1	1

- 列（蕴含项）支配：
- 如果对所有 k，都有 $a_{ki} \geq a_{kj}$
  - 移除列 j（被支配的）
- 被支配的蕴含项（j）的最小项已经被支配蕴含项（i）覆盖了

	α	β	γ	δ
000	1	0	1	0
001	0	1	1	0
101	1	1	1	0
111	1	0	0	1
110	0	0	1	1

- 行（最小项）支配：
- 如果对所有 k，都有 $a_{ik} \geq a_{jk}$
  - 移除行 i（支配的）
- 当一个蕴含项覆盖了被支配的最小项，它也覆盖了支配的最小项

```
EXACT_COVER(A, x, b) {
    如果 current_estimate ≥ |b|， 则返回 b;
    对矩阵 A 进行化简并更新对应的 x;
    如果 A 没有任何行，则返回 x;
    选择一个用于分支的列 c;
    xc = x + 1;
    Ã = 删除列 c 以及与其相关的行后的 A;
    x~ = EXACT_COVER(Ã, xc, b);
    如果 |x~| < |b|， 则
        b = x~;
    xc = x;
    Ã = 删除列 c 后的 A;
    x~ = EXACT_COVER(Ã, xc, b);
    如果 |x~| < |b|， 则
        b = x~;
    返回 b;
}
```

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

变量	值
x	0
b	∞
c	
x_c	

预测

Estimate

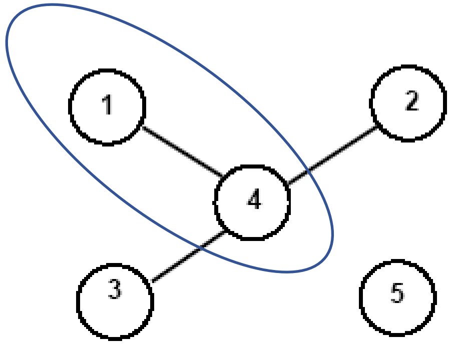
- 第四行和1，2，3是独立的
- 最大的团的节点数是2
- 预测最少需要的蕴含项是：

  最大团的结点数+已选择的基本要素数x

  即2+0=2

  返回2

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$



$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

变量	值
x	0
b	∞
c	
x_c	

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

变量	值
x	0
b	∞
c	
x_c	

缩减矩阵

Reduce matrix

$$
\begin{array}{cc}
 & \begin{array}{ccccc} & & & \surd & \end{array} \\
\mathbf{A} = & \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}
\end{array}
$$

- 进入时的x：0
- 因为第4行只有一个1，因此该1所在的第4列是基本要素，选中该列，x+1
- 移除被第4列覆盖的第4行和第5行
- 第5列是被支配的，直接移除
- 更新后的x：1
- 减少后的矩阵：

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

PS：行代表最小项，列代表蕴含项

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

变量	值
x	1
b	∞
c	
x_c	

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

变量	值
x	1
b	∞
c	A[0]
x_c	

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad \tilde{\mathbf{A}} = \begin{array}{c} \begin{array}{ccc} \surd & & \end{array} \\ \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \end{array}$$

EXACT_COVER(A,x,b) {

  if (current_estimate ≥ |b|) return (b);

  Reduce matrix A and update corresponding x;

  if (A has no rows) return(x);

  select a branching column c;

  x_c = x+1;

  Ã = A after deleting c and rows incident to it;

  $x^{\sim}$ = *EXACT_COVER*(Ã,x_c,*b*);

  if (| $x^{\sim}$| < |b|)

      b = $x^{\sim}$;

  x_c = x;

  Ã = A after deleting c;

  $x^{\sim}$ = *EXACT_COVER*(Ã,x_c,*b*);

  if (| $x^{\sim}$| < |b|)

      b = $x^{\sim}$;

  return(b);

}

变量	值
x	1
b	∞
c	A[0]
x_c	2

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$\tilde{A} = [11]$

变量	值
x	1
b	∞
c	A[0]
x_c	2

$A = [1\ \ 1]$

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x~ = EXACT_COVER(Ã,x_c,b);
    if ( | x~| < |b| )
            b = x~;
    x_c = x;
    Ã = A after deleting c;
    x~ = EXACT_COVER(Ã,x_c,b);
    if ( | x~| < |b| )
            b = x~;
    return(b);
}
```

变量	值
x	1
b	∞
c	A[0]
x_c	2

变量	值
x	2
b	∞
c	
x_c	

$A = [1\ \ 1]$

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x~ = EXACT_COVER(Ã,x_c,b);
    if ( | x~| < |b| )
            b = x~;
    x_c = x;
    Ã = A after deleting c;
    x~ = EXACT_COVER(Ã,x_c,b);
    if ( | x~| < |b| )
            b = x~;
    return(b);
}
```

变量	值
x	1
b	∞
c	A[0]
x_c	2

变量	值
x	2
b	∞
c	
x_c	

缩减矩阵

Reduce matrix

- 进入时的x：2
- 第1列是被支配的，直接移除
- 因为第1行只有一个1，因此该1所在的第2列是基本要素，选中该列，x+1
- 更新后的x：3
- 减少后的矩阵：

$$\begin{array}{cc} & \surd \\ A = [\cancel{1} & \cancel{1}] \end{array}$$

$$A = []$$

PS：行代表最小项，列代表蕴含项

$A = []$

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x~ = EXACT_COVER(Ã,x_c,b);
    if ( | x~| < |b| )
            b = x~;
    x_c = x;
    Ã = A after deleting c;
    x~ = EXACT_COVER(Ã,x_c,b);
    if ( | x~| < |b| )
            b = x~;
    return(b);
}
```

变量	值
x	1
b	∞
c	A[0]
x_c	2

变量	值
x	3
b	∞
c	
x_c	

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

```
EXACT_COVER(A,x,b) {
   if (current_estimate ≥ |b|) return (b);
   Reduce matrix A and update corresponding x;
   if (A has no rows) return(x);
   select a branching column c;
   x_c = x+1;
   Ã = A after deleting c and rows incident to it;
   x˜ = EXACT_COVER(Ã,x_c,b);
   if ( | x˜| < |b| )
            b = x˜;
   x_c = x;
   Ã = A after deleting c;
   x˜ = EXACT_COVER(Ã,x_c,b);
   if ( | x˜| < |b| )
            b = x˜;
   return(b);
}
```

变量	值
x	1
b	∞
c	A[0]
x_c	2
$\tilde{x}$	3

变量	值
x	3
b	∞
c	
x_c	

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

EXACT_COVER(A,x,b) {

if (current_estimate ≥ |b|) return (b);

Reduce matrix A and update corresponding x;

if (A has no rows) return(x);

select a branching column c;

x_c = x+1;

Ã = A after deleting c and rows incident to it;

$x^{\sim}$ = *EXACT_COVER(Ã,x_c,b);*

if (| $x^{\sim}$| < |b|)

b = $x^{\sim}$;

x_c = x;

Ã = A after deleting c;

$x^{\sim}$ = *EXACT_COVER(Ã,x_c,b);*

if (| $x^{\sim}$| < |b|)

b = $x^{\sim}$;

return(b);

}

变量	值
x	1
b	3
c	A[0]
x_c	2
$x^{\sim}$	3

变量	值
x	3
b	∞
c	
x_c	

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad \tilde{\mathbf{A}} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

变量	值
x	1
b	3
c	A[0]
x_c	1
$x^{\sim}$	3

变量	值
x	3
b	∞
c	
x_c	

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad \tilde{\mathbf{A}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

EXACT_COVER(A,x,b) {

if (current_estimate ≥ |b|) return (b);

Reduce matrix A and update corresponding x;

if (A has no rows) return(x);

select a branching column c;

x_c = x+1;

Ã = A after deleting c and rows incident to it;

$x^{\sim}$ = *EXACT_COVER(Ã,x_c,b);*

if (| $x^{\sim}$| < |b|)

b = $x^{\sim}$;

x_c = x;

Ã = A after deleting c;

$x^{\sim}$ = *EXACT_COVER(Ã,x_c,b);*

if (| $x^{\sim}$| < |b|)

b = $x^{\sim}$;

return(b);

}

变量	值
x	1
b	3
c	A[0]
x_c	1
$x^{\sim}$	3

变量	值
x	3
b	∞
c	
x_c	

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

EXACT_COVER(A,x,b) {

if (current_estimate ≥ |b|) return (b);

Reduce matrix A and update corresponding x;

if (A has no rows) return(x);

select a branching column c;

$x_c = x+1$;

$\tilde{A}$ = A after deleting c and rows incident to it;

$x^{\sim}$ = *EXACT_COVER*($\tilde{A}$, x_c, *b*);

if ($|x^{\sim}| < |b|$)

$b = x^{\sim}$;

$x_c = x$;

$\tilde{A}$ = A after deleting c;

$x^{\sim}$ = *EXACT_COVER*($\tilde{A}$, x_c, *b*);

if ($|x^{\sim}| < |b|$)

$b = x^{\sim}$;

return(b);

}

变量	值
x	1
b	3
c	A[0]
x_c	1
$x^{\sim}$	3

变量	值
x	3
b	∞
c	
x_c	

变量	值
x	1
b	3
c	
x_c	

预测

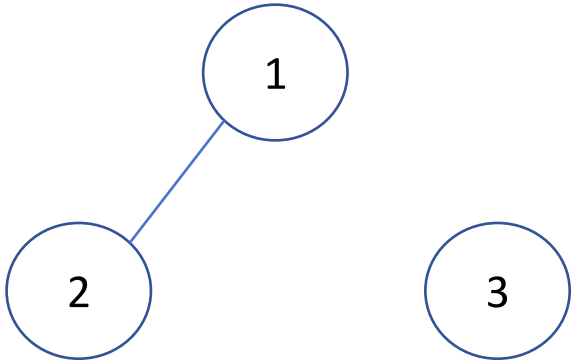
Estimate

- 第2行和第1行是独立的
- 最大的团的节点数是2
- 预测最少需要的蕴含项是：

  最大团的结点数+已选择的基本要素数

  即2+1=3

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$



$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

变量	值
x	1
b	3
c	A[0]
x_c	1
$x^{\sim}$	3

变量	值
x	3
b	∞
c	
x_c	

变量	值
x	1
b	3
c	
x_c	

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

变量	值
x	1
b	3
c	A[0]
x_c	1
$\tilde{x}$	3

变量	值
x	3
b	∞
c	
x_c	

变量	值
x	1
b	3
c	
x_c	

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

```
EXACT_COVER(A,x,b) {
    if (current_estimate ≥ |b|) return (b);
    Reduce matrix A and update corresponding x;
    if (A has no rows) return(x);
    select a branching column c;
    x_c = x+1;
    Ã = A after deleting c and rows incident to it;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    x_c = x;
    Ã = A after deleting c;
    x˜ = EXACT_COVER(Ã,x_c,b);
    if ( | x˜| < |b| )
            b = x˜;
    return(b);
}
```

变量	值
x	1
b	3
c	A[0]
x_c	1
$x^{\sim}$	3

变量	值
x	3
b	∞
c	
x_c	

变量	值
x	1
b	3
c	
x_c	

随堂作业

```
EXACT_COVER(A, x, b) {
  如果 current_estimate ≥ |b|，则返回 b；
  对矩阵 A 进行化简并更新对应的 x；
  如果 A 没有任何行，则返回 x；
  选择一个用于分支的列 c；
  xc = x + 1；
  Ã = 删除列 c 以及与其相关的行后的 A；
  x~ = EXACT_COVER(Ã, xc, b)；
  如果 |x~| < |b|，则
    b = x~；
  xc = x；
  Ã = 删除列 c 后的 A；
  x~ = EXACT_COVER(Ã, xc, b)；
  如果 |x~| < |b|，则
    b = x~；
  返回 b；
}
```

请求出以下布尔函数的最小覆盖：

F=A'B'C'D'+A'BC'D'+A'BC'D+A'BCD+A'B'CD+A'B'CD'

	0*00	010*	01*1	0*11	001*	00*0
0000	1	0	0	0	0	1
0100	1	1	0	0	0	0
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

质蕴含项表

	0*00	010*	01*1	0*11	001*	00*0
0000	1	0	0	0	0	1
0100	1	1	0	0	0	0
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项：

丢掉第1列

x=0 → x=0
b=∞ → b=∞

	0*00	010*	01*1	0*11	001*	00*0
0000	1	0	0	0	0	1
0100	1	1	0	0	0	0
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项:
010*
00*0

第2、6列是必要质蕴含项，x=x+2

x=0
b=∞

→ x=2
b=∞

		√				√
	0*00	010*	01*1	0*11	001*	00*0
0000	1	0	0	0	0	1
0100	1	1	0	0	0	0
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项:
010*
00*0

第3、5列被支配，删除

x=2
b=∞

x=2
b=∞

	0*00	010* √	01*1	0*11	001*	00*0 √
0000	1	0	0	0	0	1
0100	1	1	0	0	0	0
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项:
010*
00*0
0*11
目前最优解:
010*
00*0
0*11

第4列是必要质蕴含项，x=x+1

$x=2$
$b=\infty$ → $x=3$
$b=3$

	√ 0*00	010*	01*1	√ 0*11	001*	√ 00*0
0000	1	0	0	0	0	1
0100	1	1	0	0	0	0
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项：
0*00
目前最优解：
010*
00*0
0*11

留下第1列，x=x+1

x=0
b=3

$\longrightarrow$

x=1
b=3

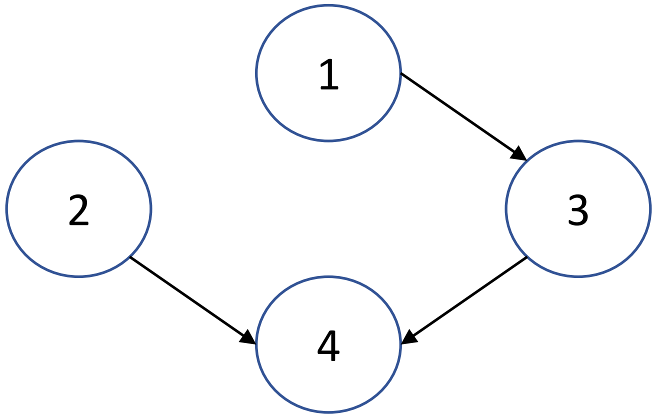
	√ 0*00	010*	01*1	0*11	001*	00*0
~~0000~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~
~~0100~~	~~1~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项：
0*00
目前最优质蕴含项：
010*
00*0
0*11

最大团的结点数为2，x=x+2

x=1 → x=3
b=3 → b=3

	010*	01*1	0*11	001*	00*0
0111	0	1	1	0	0
0011	0	0	1	1	0
0010	0	0	0	1	1
0101	1	1	0	0	0

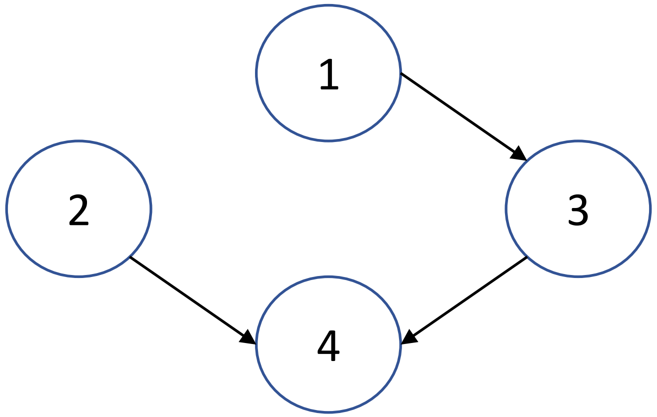


选中的质蕴含项:
0*00
目前最优质蕴含项:
010*
00*0
0*11

3=目前求出的最优解3，直接结束

x=3
b=3

	010*	01*1	0*11	001*	00*0
0111	0	1	1	0	0
0011	0	0	1	1	0
0010	0	0	0	1	1
0101	1	1	0	0	0



如果你先处理留下第一列

选中的质蕴含项：
0*00

留下第1列，x=x+1

$x=0$, $b=\infty$ $\longrightarrow$ $x=1$, $b=\infty$

	0*00 √	010*	01*1	0*11	001*	00*0
~~0000~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~
~~0100~~	~~1~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项:
0*00

第2、6列被支配，删除

x=1
b=∞

x=1
b=∞

	√ 0*00	010*	01*1	0*11	001*	00*0
~~0000~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~
~~0100~~	~~1~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项：
0*00
01*1
001*
目前最优解：
0*00
01*1
001*

第3、5列是必要质蕴含项，x=x+2

$x=1$
$b=\infty$ $\longrightarrow$ $x=3$
$b=3$

	√		√		√	
	0*00	010*	01*1	0*11	001*	00*0
0000	1	0	0	0	0	1
0100	1	1	0	0	0	0
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项:

目前最优解:
0*00
01*1
001*

丢掉第1列

x=0
b=3

x=0
b=3

	0*00	010*	01*1	0*11	001*	00*0
0000	1	0	0	0	0	1
0100	1	1	0	0	0	0
0111	0	0	1	1	0	0
0011	0	0	0	1	1	0
0010	0	0	0	0	1	1
0101	0	1	1	0	0	0

选中的质蕴含项:

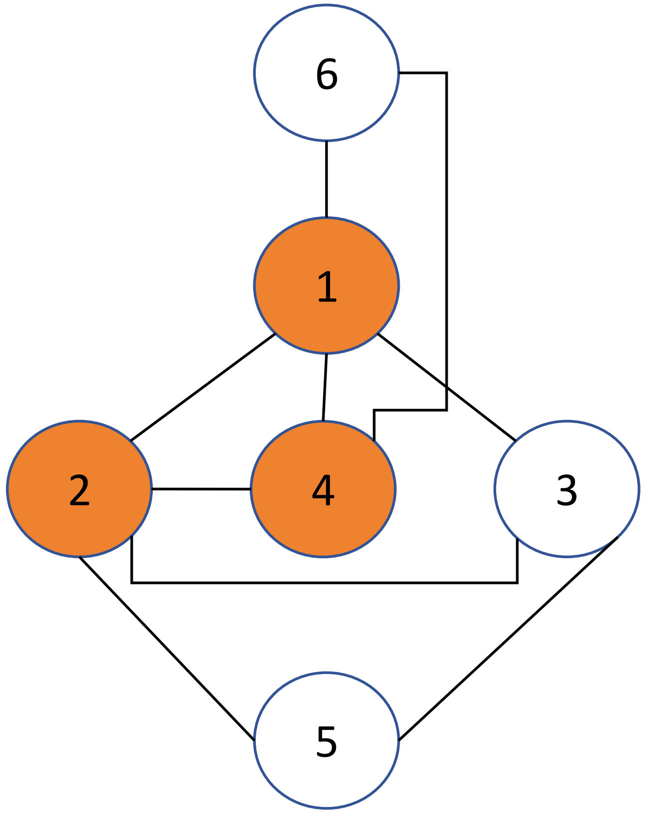
目前最优解:
0*00
01*1
001*

找到一个团的结点数为3, x=x+3

x=0
b=3

x=3
b=3

	010*	01*1	0*11	001*	00*0
0000	0	0	0	0	1
0100	1	0	0	0	0
0111	0	1	1	0	0
0011	0	0	1	1	0
0010	0	0	0	1	1
0101	1	1	0	0	0



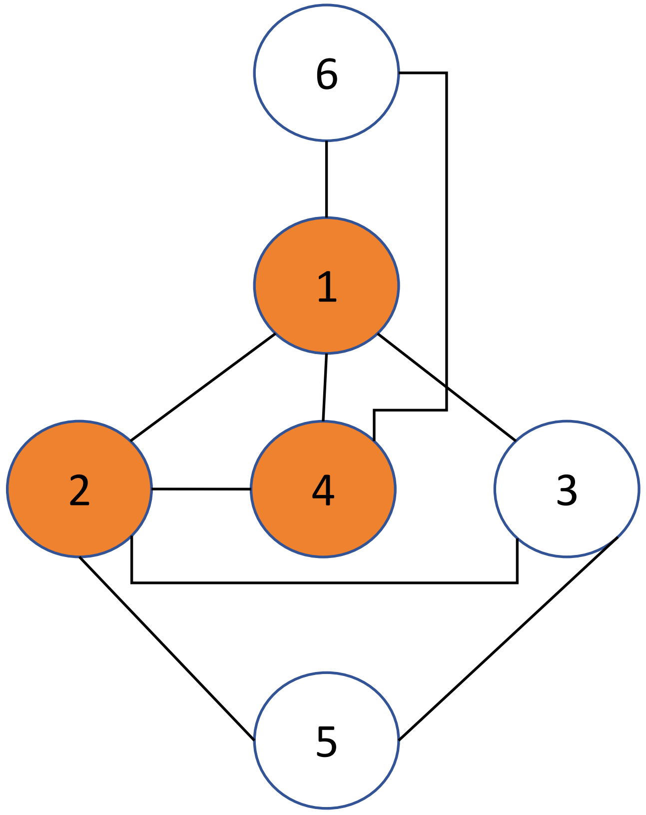
选中的质蕴含项:

目前最优质蕴含项:
0*00
01*1
001*

3=目前求出的最优解3，直接结束

x=3
b=3

	010*	01*1	0*11	001*	00*0
0000	0	0	0	0	1
0100	1	0	0	0	0
0111	0	1	1	0	0
0011	0	0	1	1	0
0010	0	0	0	1	1
0101	1	1	0	0	0



Espresso-exact

Espresso-exact

- 精确的两级逻辑最小化器
- 利用分支定界算法
- 实现时用到了蕴含项表
- 对于大多数基准都能非常高效地得到最佳解

最低覆盖率的早期方法

Minimum cover early methods

- 简化表格
  - 迭代地识别必要元素，将它们保存在覆盖中。
  - 移除已覆盖的最小项。
- Petrick 方法
  - 以积之和（POS）形式写覆盖子句
  - 将积之和形式展开为和之积（SOP）的形式
  - 选择最小的立方项
  - 注意：展开子句的成本是指数级的

例子

Example

- $f = a'b'c' + a'b'c + ab'c + abc + abc'$
- 蕴含项表:

	abc	f
α	00*	1
β	*01	1
γ	1*1	1
δ	11*	1

000： (α)

001： $(\alpha + \beta)$

101： $(\beta + \gamma)$

111： $(\gamma + \delta)$

110： (δ)

- pos形式：
  - $(\alpha)(\alpha + \beta)(\beta + \gamma)(\gamma + \delta)(\delta) = 1$
- 展开为sop形式:
  - $\alpha\beta\delta + \alpha\gamma\delta = 1$

$$(\alpha)\ (\alpha + \beta)\ \ (\beta + \gamma)\ (\gamma + \delta)\ (\delta)$$

$$=(\alpha + \alpha\beta)\ (\beta\ \gamma + \beta\ \delta\ +\ \gamma\ +\ \gamma\ \delta)\ \delta$$

$$=(\alpha)(\gamma+\beta\ \delta\ +\ \gamma\ \delta)\ \delta$$

$$=\ \alpha\delta\ (\gamma+\beta\ \delta)$$

$$=\ \alpha\gamma\delta+\ \alpha\ \beta\ \delta$$

例子

Example

- $f = a'b'c' + a'b'c + ab'c + abc + abc'$
- 蕴含项表:

	abc	f
α	00*	1
β	*01	1
γ	1*1	1
δ	11*	1

000： (α)

001： (α + β)

101： (β + γ)

111： (γ + δ)

110： (δ)

- pos形式：
  - (α) (α + β) (β + γ) (γ + δ) (δ) = 1
- 展开为sop形式:
  - αβδ + αγδ = 1
- 解:

{ α β δ } $f = a'b' + b'c + ab$

或 { α γ δ } $f = a'b' + ac + ab$

用ILP形式化

- 将表格视为布尔矩阵: A
- 选择一个布尔向量: x
- 确定 x 使得:
  - A * x ≥ 1
  - 选择足够多的列来覆盖所有行
- 最小化 x 的范数（即最小化x中1的数量）

	α	β	γ	δ
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

$$\longrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathrm{X} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} >= \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

矩阵表示

Matrix representation

- 布尔变量：

α、β、γ、δ

- 最小化：

α+β+γ+δ

- 约束：

α*1+β*0+γ*0+δ*0>=1

α*1+β*1+γ*0+δ*0>=1

α*0+β*1+γ*1+δ*0>=1

α*0+β*0+γ*1+δ*1>=1

α*0+β*0+γ*0+δ*1>=1

	α	β	γ	δ
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

$$\longrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{X} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} >= \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

例子

Example

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix} >= \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$