# 形式化方法大作业

# ScissorSAT——基于割集并行的SAT求解策略

PB22111649 何跃强

2025-6-27

## 1. SAT求解器背景

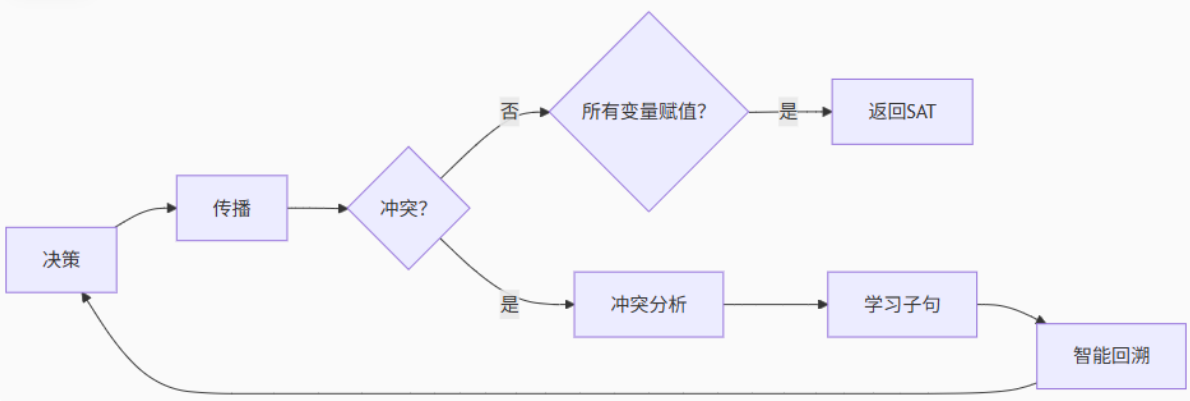
### 1.1 SAT问题概述

布尔可满足性问题（Boolean Satisfiability Problem, SAT）是计算机科学的核心问题，询问是否存在一组布尔变量赋值使得给定逻辑公式为真。其重要性在于：

- 首个被证明的NP完全问题（Cook-Levin定理）
- 广泛应用在硬件验证、软件测试、AI规划等领域
- 现代求解器可处理百万变量级工业问题

### 1.2 CDCL算法原理

冲突驱动子句学习（Conflict-Driven Clause Learning）是现代SAT求解器的核心算法：




关键创新：

- 两文字监视：高效传播（BCP）
- VSIDS启发式：动态变量选择
- 学习子句管理：LBD质量评估

## 2. 前人对SAT求解器并行的求解方案

### 2.1、并行SAT求解框架的主要类型

| 类别 | 框架描述 | 代表系统/研究 |
| --- | --- | --- |
| **Portfolio-based** | 多个不同配置的CDCL求解器实例并行运行，竞争性地解决同一问题 | *ManySAT*, *Plingeling* |
| **Divide-and-conquer** | 将原问题拆分成多个子问题分配给不同核，解空间划分明确 | *PSATO*, *PaMiraXT*, *Treengeling* |

| 类别 | 框架描述 | 代表系统/研究 |
| --- | --- | --- |
| **Hybrid methods** | 结合Portfolio与Divide方法，部分共享搜索信息 | *Parallel CryptoMiniSat* |
| **Clause-sharing** | 多线程间周期性共享learned clauses | *ManySAT*, *Plingeling* |
| **Lookahead-based** | 利用静态预处理分析做问题划分，提升分支质量 | *Cube-and-Conquer*, *March_cc* |

# 2.2、各类并行SAT求解器框架的评估

## 2.2.1. Portfolio-based 框架

- **优点**:
  - 零通信开销，线程间独立运行，适合异构环境。
  - 对不同问题类型有稳健表现。
  - 易于实现，无需重构求解器核心。
- **缺点**:
  - 资源利用不充分，线程间冗余计算严重。
  - 无法保证提升所有案例的求解效率。
  - 缺乏协作机制，对大规模问题不够友好。

## 2.2.2. Divide-and-conquer 框架

- **优点**:
  - 理论上具备最好的可扩展性和速度加速比。
  - 可直接映射到多核、集群甚至超算节点。
  - 利用cutset、cube等结构信息可控性强。
- **缺点**:
  - 子问题划分难度大，不均衡时会造成load imbalance。
  - 解空间划分需要额外预处理。
  - 子问题之间结果回传合并逻辑复杂。

## 2.2.3. Hybrid 框架

- **优点**:
  - 兼顾竞争与协作，部分共享信息提升效率。
  - 平衡了冗余和通信的开销。
- **缺点**:
  - 实现复杂，需要合理的调度与同步机制。
  - clause-sharing过多可能导致污染与性能下降。

### 2.2.4. Clause-sharing 策略

- **优点**：
  - 有效加速分支裁剪，避免重复搜索。
  - 在结构规则的实例中非常高效（如硬件验证）。
- **缺点**：
  - 冗余子句污染其他线程的搜索路径。
  - 通信频率与策略调节极为关键（需设置LBD阈值等）。

### 2.2.5. Lookahead/Cube-and-Conquer 框架

- **优点**：
  - 适用于结构化强、约束密集的问题。
  - 解空间分区合理，提升CDCL求解质量。
- **缺点**：
  - lookahead耗时严重，预处理成本高。
  - 对非结构化问题表现可能不佳。

## 2.3、评估总结表格

| 方向 | 可扩展性 | 实现难度 | 通信开销 | 鲁棒性 | 适用场景 |
|---|---|---|---|---|---|
| Portfolio | 中 | 低 | 无 | 高 | 各类SAT |
| Divide&Conq. | 高 | 高 | 中 | 中 | 大规模SAT |
| Hybrid | 高 | 高 | 中 | 高 | 通用型 |
| Clause-sharing | 中 | 中 | 高 | 中 | 工业实例 |
| Lookahead/C&C | 高 | 高 | 低 | 中 | 带结构问题 |

## 2.4、我们提出的新方法

上述方法经常需要进行通信，然而独立性的运行可以极大的减少通信的时间，因此我提出了基于割集的CDCL并行方法（**ScissorSAT**），它很好的解决了通信量较大的问题并展现出了很好的并行拓展性。

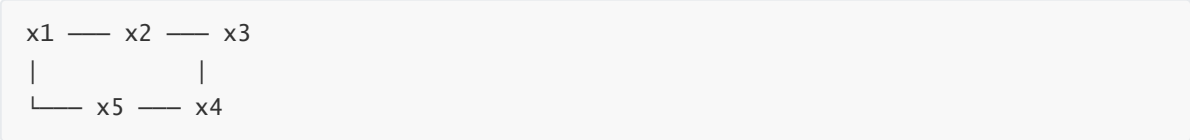
# 3. 基于割集的CDCL并行方法

## 3.1 割集理论在SAT中的应用

**割集定义**：在变量共现图中，割集是移除后使图分裂为多个连通分量的最小顶点集

**数学表示**：
设SAT实例对应图G=(V,E)，割集C⊆V满足：
$G\backslash C = G_1 \cup G_2 \cup \ldots \cup G_k$ ($k \geq 2$)

**示例**：
考虑CNF：`(x1∨x2) ∧ (¬x2∨x3) ∧ (x4∨x5) ∧ (¬x1∨¬x5)`
变量图：




割集C={x2, x5}，移除后得两个分量：

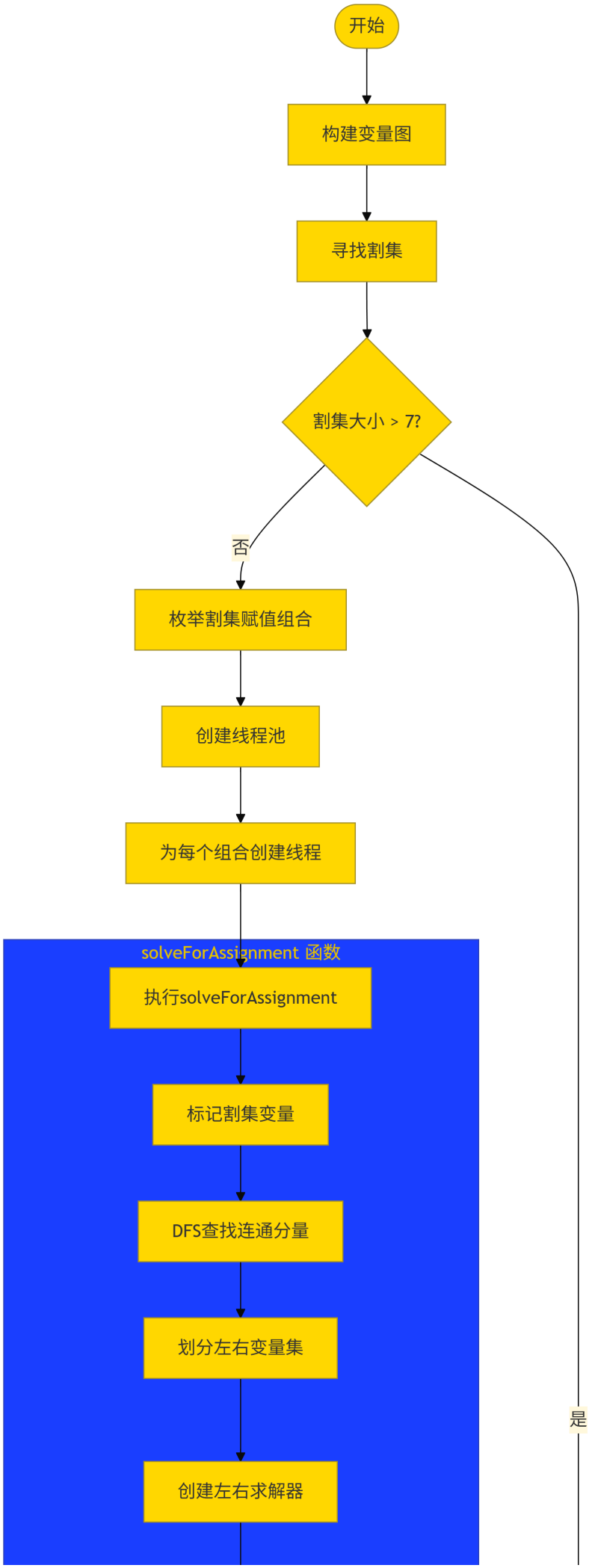
- $G_1$: {x1, x3}
- $G_2$: {x4}

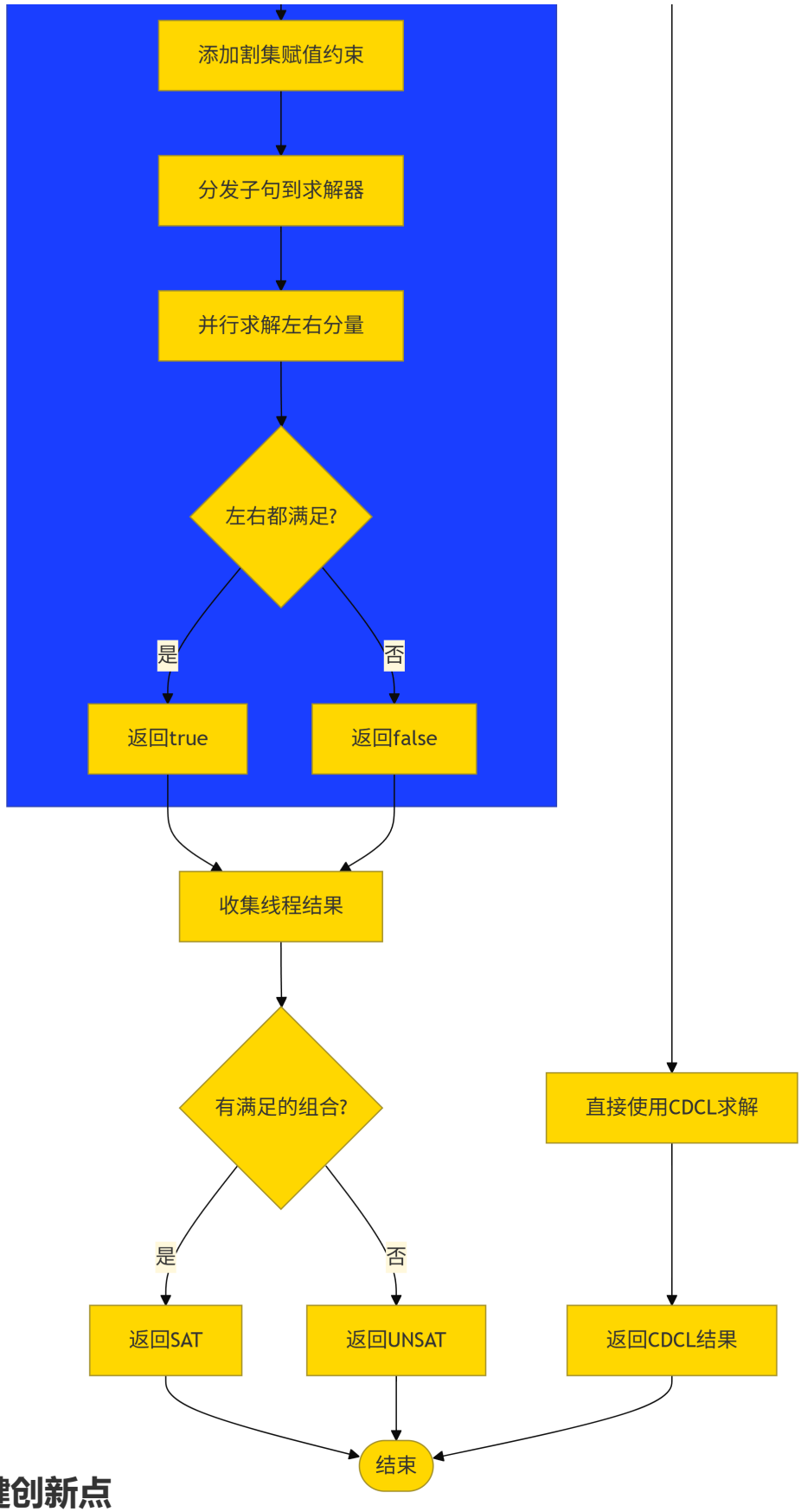
## 3.2 并行策略设计

### 3.2.1 基本并行框架

**步骤如下**：

1. 构造变量图
2. 进行割集搜索 (使用贪心算法求解)
3. 分别列举前 k 个割集元素的全部真假赋值
4. 将该赋值作为断言条件加入两个子问题
5. 传递其余非割集的子句
6. 将子问题并行进行 CDCL
7. 如果有一个 UNSAT，则分支 UNSAT
8. 如果两者均 SAT，则返回 SAT


开始
构建变量图
寻找割集
割集大小 > 7?
否
是
枚举割集赋值组合
创建线程池
为每个组合创建线程
solveForAssignment 函数
执行solveForAssignment
标记割集变量
DFS查找连通分量
划分左右变量集
创建左右求解器




## 3.3 关键创新点

1. **分层并行**：割集枚举(外层) + 分量求解(内层)
2. **冲突驱动线程管理**：任一UNSAT立即终止分支

# 4. 实现与优化

## 4.1 代码修改设计

我们使用 Minisat : MiniSat Page 的源代码进行修改，经过设计我们进行以下的修改：

```
minisat/
├── core/
|   ├── Solver.cc             <-- 修改：添加 CDCL 并行调用支持
|   ├── Solver.h              <-- 修改：添加分支子问题处理接口
├── parallel/
|   ├── CutsetSplitter.cc     <-- 新增：图构建、cutset 查找与分割 CNF
|   ├── CutsetSplitter.h      <-- 新增：声明接口
├── utils/
|   ├── Graph.cc              <--新增：变量图构建（变量邻接图）
|   ├── Graph.h
├── main/
|   ├── main_parallel.cc      <-- 新增：基于 cutset 并行 CDCL 启动程序
├── Makefile                  <-- 修改：加入 parallel/ 与 utils/ 文件编译项
```

## 4.2 代码具体实现

由于篇幅限制，我们只考虑对具体函数实现进行列举

`CutsetSplitter.h`

```
// parallel/CutsetSplitter.h
#ifndef CUTSET_SPLITTER_H
#define CUTSET_SPLITTER_H
#include "minisat/core/SolverTypes.h"
#include "minisat/utils/Graph.h"

using namespace Minisat;

class CutsetSplitter {
public:
    CutsetSplitter(const std::vector<std::vector<Lit>>& clauses, int num_vars);//
对总子句进行分解与并行求解
    bool solveByCutset();
private:
    std::vector<std::vector<Lit>> clauses;
    int num_vars;
    bool solveForAssignment(const std::vector<int>& cutset, const
std::vector<bool>& assignment , Graph& g);//对分任务进行调度和求解
};

#endif
```

`Graph.h`

```
// utils/Graph.h
#ifndef GRAPH_H
#define GRAPH_H
```

```
#include <vector>
#include <unordered_set>

class Graph {
public:
    Graph(int num_vars);
    void addEdge(int v1, int v2);
    std::vector<int> findCutset(int max_cutsize = 7); // 返回一个小割集
    bool isConnected(const std::vector<bool>& is_cut); // 检查割集是否断开图的连接
    std::vector<std::unordered_set<int>> adj;
private:
    // std::vector<std::unordered_set<int>> adj;
};

#endif
```

总体步骤：

`main_parallel.cc`

```
/**************************************************************************************************
*********[main_parallel.cc]
    Based on MiniSat Main.cc. Modified to support cutset-based parallel CDCL.
**************************************************************************************************
*****************/

#include <errno.h>
#include <zlib.h>
#include <thread>
#include <vector>

#include "minisat/utils/System.h"
#include "minisat/utils/ParseUtils.h"
#include "minisat/utils/Options.h"
#include "minisat/core/Dimacs.h"
#include "minisat/core/Solver.h"

#include "minisat/parallel/CutsetSplitter.h"

using namespace Minisat;

static Solver* solver;

// 中断处理
static void SIGINT_interrupt(int) { solver->interrupt(); }
static void SIGINT_exit(int) {
    printf("\n*** INTERRUPTED ***\n");
    if (solver->verbosity > 0){
        solver->printStats();
        printf("\n*** INTERRUPTED ***\n"); }
    _exit(1);
}

int main(int argc, char** argv)
```

```
{
    try {
        printf("Cutset-based Parallel CDCL Solver\n");
        printf("Based on MiniSat\n");
        setUsageHelp("USAGE: %s [options] <input-file> <result-output-file>\n\n"
                     "  where input may be either in plain or gzipped
DIMACS.\n");
        setX86FPUPrecision();

        IntOption    verb   ("MAIN", "verb",   "Verbosity level (0=silent,
1=some, 2=more).", 1, IntRange(0, 2));
        IntOption    cpu_lim("MAIN", "cpu-lim","Limit on CPU time allowed in
seconds.\n", 0, IntRange(0, INT32_MAX));
        IntOption    mem_lim("MAIN", "mem-lim","Limit on memory usage in
megabytes.\n", 0, IntRange(0, INT32_MAX));
        BoolOption   strictp("MAIN", "strict", "Validate DIMACS header during
parsing.", false);

        parseOptions(argc, argv, true);

        Solver S;
        double initial_time = cpuTime();

        S.verbosity = verb;
        solver = &S;

        sigTerm(SIGINT_exit);

        if (cpu_lim != 0) limitTime(cpu_lim);
        if (mem_lim != 0) limitMemory(mem_lim);

        if (argc == 1)
            printf("Reading from standard input... Use '--help' for help.\n");

        gzFile in = (argc == 1) ? gzdopen(0, "rb") : gzopen(argv[1], "rb");
        if (in == NULL){
            printf("ERROR! Could not open file: %s\n", argc == 1 ? "<stdin>" :
argv[1]);
            exit(1);
        }

        if (S.verbosity > 0){
            printf("============================[ Problem Statistics
]=============================\n");
            printf("|
                |\n");
        }

        parse_DIMACS(in, S, (bool)strictp);
        gzclose(in);

        FILE* res = (argc >= 3) ? fopen(argv[2], "wb") : NULL;

        if (S.verbosity > 0){
            printf("|  Number of variables:  %12d
        |\n", S.nVars());
```

```
            printf("|  Number of clauses:    %12d
        |\n", S.nClauses());
        }

        double parsed_time = cpuTime();
        if (S.verbosity > 0){
            printf("|  Parse time:           %12.2f s
          |\n", parsed_time - initial_time);
            printf("|
                |\n");
        }

        sigTerm(SIGINT_interrupt);
        //
printf("================================================================================
======\n");

        if (!S.simplify()){
            if (res != NULL) fprintf(res, "UNSAT\n"), fclose(res);
            if (S.verbosity > 0){

 printf("================================================================================
=======\n");
                printf("Solved by unit propagation\n");
                S.printStats();
                printf("\n");
            }
            printf("UNSATISFIABLE\n");
            exit(20);
        }
        printf("Simplification complete.\n");

        // ------------------- 构建 CutsetSplitter 并求解 -------------------
        std::vector<std::vector<Lit>> clauses;
        for (int i = 0; i < S.clauses.size(); i++) {
            Clause& c = S.ca[S.clauses[i]];
            std::vector<Lit> cl;
            for (int j = 0; j < c.size(); ++j)
                cl.push_back(c[j]);
            clauses.push_back(cl);
        }
        printf("Total clauses: %zu\n", clauses.size());

        CutsetSplitter splitter(clauses, S.nVars());
        // 打印splitter的所有信息
        if (S.verbosity > 0) {

 printf("================================================================================
=======\n");
            printf("Cutset Splitter Information:\n");
            printf("Number of variables: %d\n", S.nVars());
            printf("Number of clauses: %zu\n", clauses.size());
        }

        printf("Starting cutset-based parallel CDCL...\n");
        bool result = splitter.solveByCutset();
```

```
        if (S.verbosity > 0) {

 printf("================================================================================
=======\n");
            S.printStats();
            printf("\n");
        }

        printf(result ? "SATISFIABLE\n" : "UNSATISFIABLE\n");

        if (res != NULL) {
            fprintf(res, result ? "SAT\n" : "UNSAT\n");
            fclose(res);
        }

#ifdef NDEBUG
        exit(result ? 10 : 20);
#else
        return result ? 10 : 20;
#endif

    } catch (OutOfMemoryException&) {

 printf("================================================================================
=======\n");
        printf("INDETERMINATE\n");
        exit(0);
    }
}
```

# 5. 实验结果

## 5.1 数据集选择

我们采用python自带的随机SAT式子生成器进行求解：

```
# 使用CNFgen生成随机问题
pip install cnfgen
cnfgen randkcnf 3 100 420 > random_3sat.cnf
```

同时考虑到我们对最大割集的限制数为7（由于计算机的物理核心限制），经过统计，一般考虑割集占所有点的10%。于是我们选择了100个变量元素进行考虑。

## 5.2 运行命令

运行基础minisat命令：

```
./build/release/bin/minisat random_3sat.cnf
```

运行并行命令：

```
./build/release/bin/minisat_parallel random_3sat.cnf
```

## 5.3 部分实验结果展示

数据一：

传统CDCL方法：

```
(AI) (base) hyq@hyq:~/Formal/minisat$ ./build/release/bin/minisat random_3sat.cnf
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:          100                                          |
|  Number of clauses:            420                                          |
|  Parse time:                  0.00 s                                        |
|  Eliminated clauses:          0.00 Mb                                       |
|  Simplification time:         0.00 s                                        |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|       100 |      95      412     1262 |      151      100      7 |  0.010 % |
|       250 |      95      412     1262 |      166       89      6 |  0.010 % |
|       475 |      93      395     1210 |      182      103      6 |  2.030 % |
===============================================================================
restarts              : 5
conflicts             : 522            (146918 /sec)
decisions             : 602            (0.00 % random) (169434 /sec)
propagations          : 11982          (3372361 /sec)
conflict literals     : 3129           (22.74 % deleted)
Memory used           : 5.84 MB
CPU time              : 0.003553 s

UNSATISFIABLE
```

基于割集的并行CDCL方法：

```
(AI) (base) hyq@hyq:~/Formal/minisat$ ./build/release/bin/minisat_parallel random_3sat.cnf
Cutset-based Parallel CDCL Solver
Based on MiniSat
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:          100                                          |
|  Number of clauses:            420                                          |
|  Parse time:                  0.00 s                                        |
|                                                                             |
Simplification complete.
Total clauses: 420
===============================================================================
Cutset Splitter Information:
Number of variables: 100
Number of clauses: 420
Starting cutset-based parallel CDCL...
===============================================================================
restarts              : 0
conflicts             : 0              (0 /sec)
decisions             : 0              (-nan % random) (0 /sec)
propagations          : 0              (0 /sec)
conflict literals     : 0              (-nan % deleted)
Memory used           : 10.62 MB
CPU time              : 0.002824 s

UNSATISFIABLE
```

数据二：

传统CDCL方法：

```
(AI) (base) hyq@hyq:~/Formal/minisat$ ./build/release/bin/minisat random_3sat.cnf
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:           100                                         |
|  Number of clauses:             420                                         |
|  Parse time:                   0.00 s                                       |
|  Eliminated clauses:           0.00 Mb                                      |
|  Simplification time:          0.00 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|       100 |      99      419     1261 |      153      100      8 |  0.010 % |
|       250 |      99      419     1261 |      168       93      7 |  0.010 % |
===============================================================================
restarts              : 3
conflicts             : 368          (119870 /sec)
decisions             : 456          (0.00 % random) (148534 /sec)
propagations          : 8546         (2783713 /sec)
conflict literals     : 2513         (19.38 % deleted)
Memory used           : 5.84 MB
CPU time              : 0.00307 s

SATISFIABLE
```

基于割集的并行CDCL方法：

```
(AI) (base) hyq@hyq:~/Formal/minisat$ ./build/release/bin/minisat_parallel random_3sat.cnf
Cutset-based Parallel CDCL Solver
Based on MiniSat
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:           100                                         |
|  Number of clauses:             420                                         |
|  Parse time:                   0.00 s                                       |
|                                                                             |
Simplification complete.
Total clauses: 420
===============================================================================
Cutset Splitter Information:
Number of variables: 100
Number of clauses: 420
Starting cutset-based parallel CDCL...
===============================================================================
restarts              : 0
conflicts             : 0            (0 /sec)
decisions             : 0            (-nan % random) (0 /sec)
propagations          : 0            (0 /sec)
conflict literals     : 0            (-nan % deleted)
Memory used           : 10.62 MB
CPU time              : 0.00276 s
```

经过大量数据统计，我们可以看到，在割集比较少的情况下，无论是否为满足，并行CDCL都是要比CDCL要好，但是加速并不明显。对于不同方面，SATISFIED加速要明显小于UNSATISFIED的结果。可以直观的分析得出原因，因为UNSATISFIED需要遍历，而这种并行很好的切割了遍历的搜索空间。

对于较少的元素，例如10个元素，或者较多的元素例如1000个元素，由于并行启动时间以及割集大小的问题并行几乎没有加速的效果，因此还需要进行进一步修改。

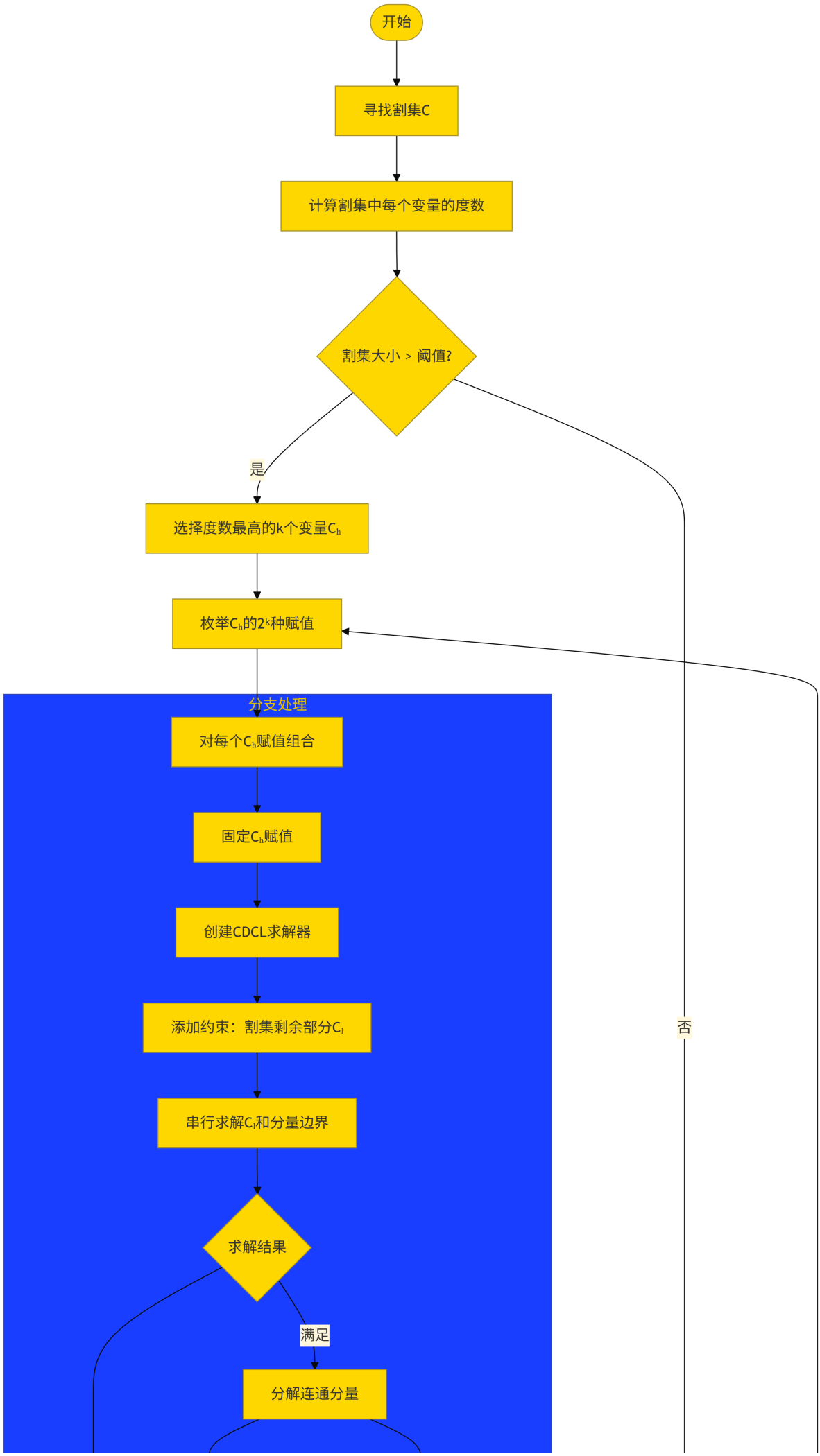
# 6. 局限性以及未来工作

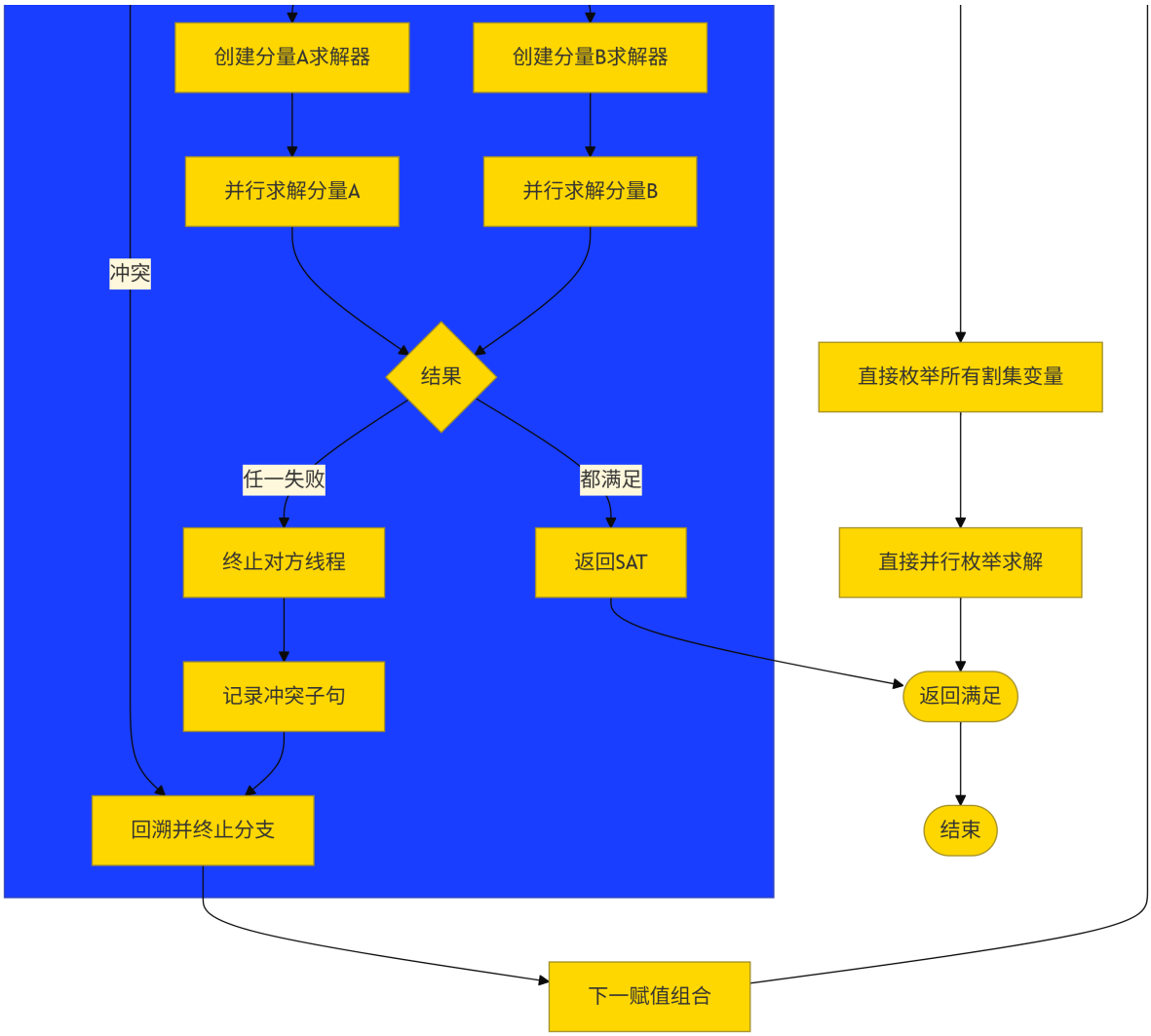
由于并行受割集影响有很大的限制，同时为了更好激发其潜能，我提出了以下几种新的idea，由于时间原因（期末考试TT）,我只提出了进一步的修改方案。

## 6.1 为了解决性能与并行度受割集影响较大的问题，我提出了这样的解决方案：

**总体思路：先求出其割集，无论割集大小，假设割集有20个元素，只选取度数最高的前5个或者前10个进行列举，然后对割集的剩余部分进行串行的CDCL求解，最后再对分割出来的两个集合进行并行CDCL求解，只要回溯到割集就判定为另一方失败并进行重启，如果两个都满足就说明可以满足。**

流程图如下：

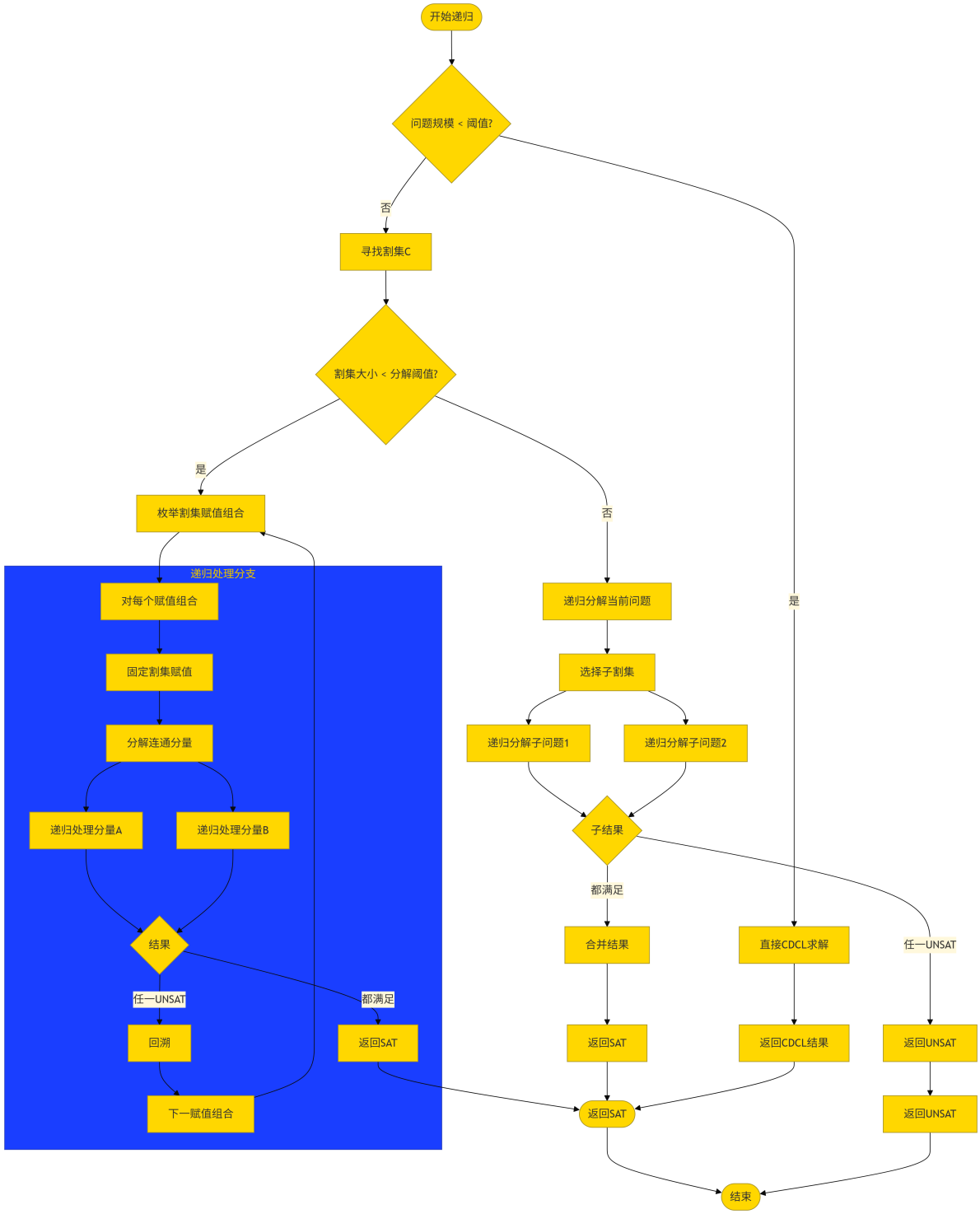

开始
寻找割集C
计算割集中每个变量的度数
割集大小 > 阈值?
是
否
选择度数最高的k个变量$C_h$
枚举$C_h$的$2^k$种赋值
分支处理
对每个$C_h$赋值组合
固定$C_h$赋值
创建CDCL求解器
添加约束：割集剩余部分$C_l$
串行求解$C_l$和分量边界
求解结果
满足
分解连通分量


创建分量A求解器
创建分量B求解器
并行求解分量A
并行求解分量B
冲突
结果
任一失败
都满足
终止对方线程
返回SAT
记录冲突子句
回溯并终止分支
直接枚举所有割集变量
直接并行枚举求解
返回满足
结束
下一赋值组合

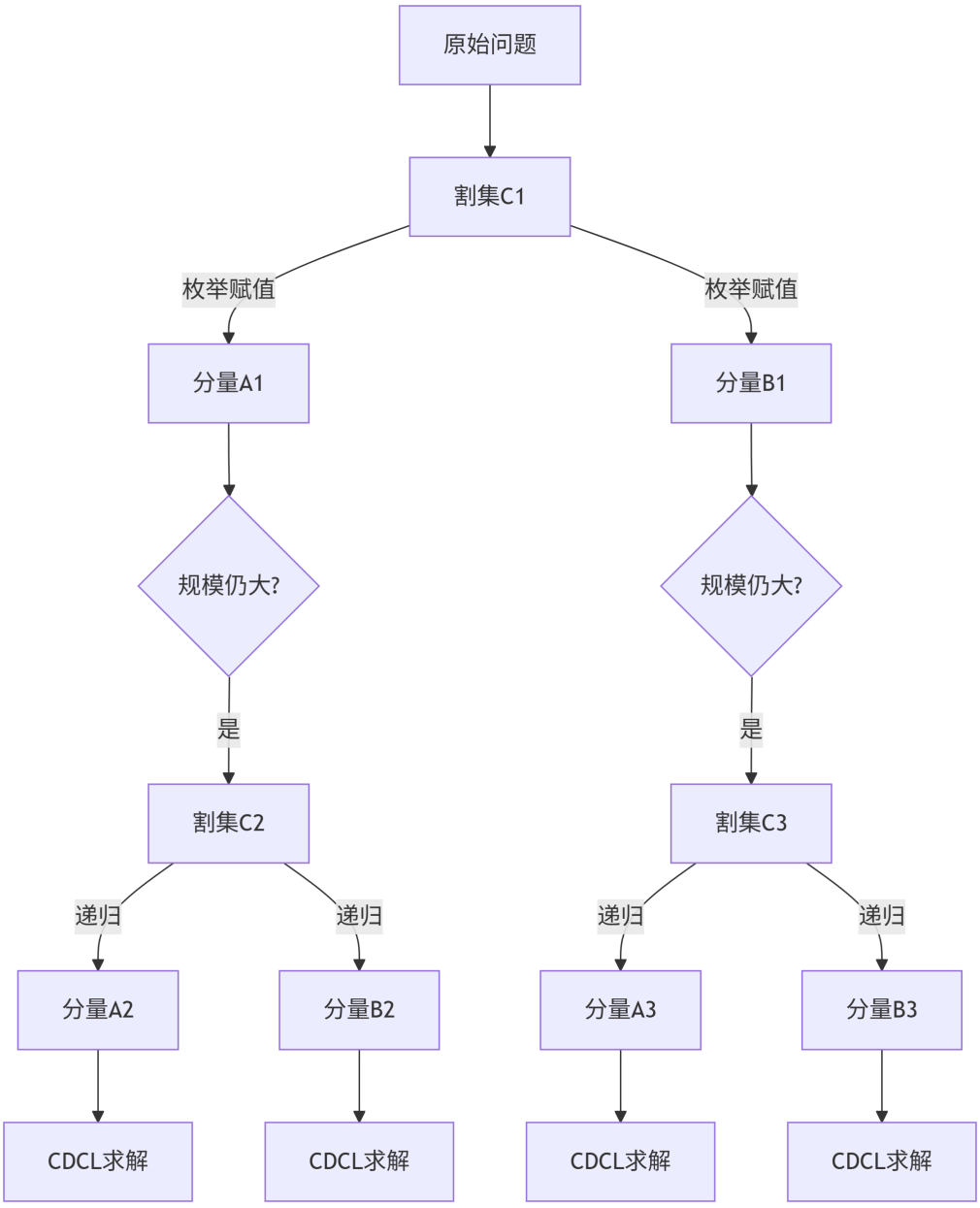
## 6.2 为了解决并行度不够的问题，我提出以分治法的思想解决这个问题

**我们发现每个整问题可以拆成多个子问题，于是我们可以对子问题进行递归，一直到计算机很容易就能求解出来的结果（例如总元素数目小于100的时候）直接用CDCL求解，并向上归纳，最后通过这样可以极大的拓展并行性，同时也预防了搜索空间指数爆炸。**

下面是流程图：

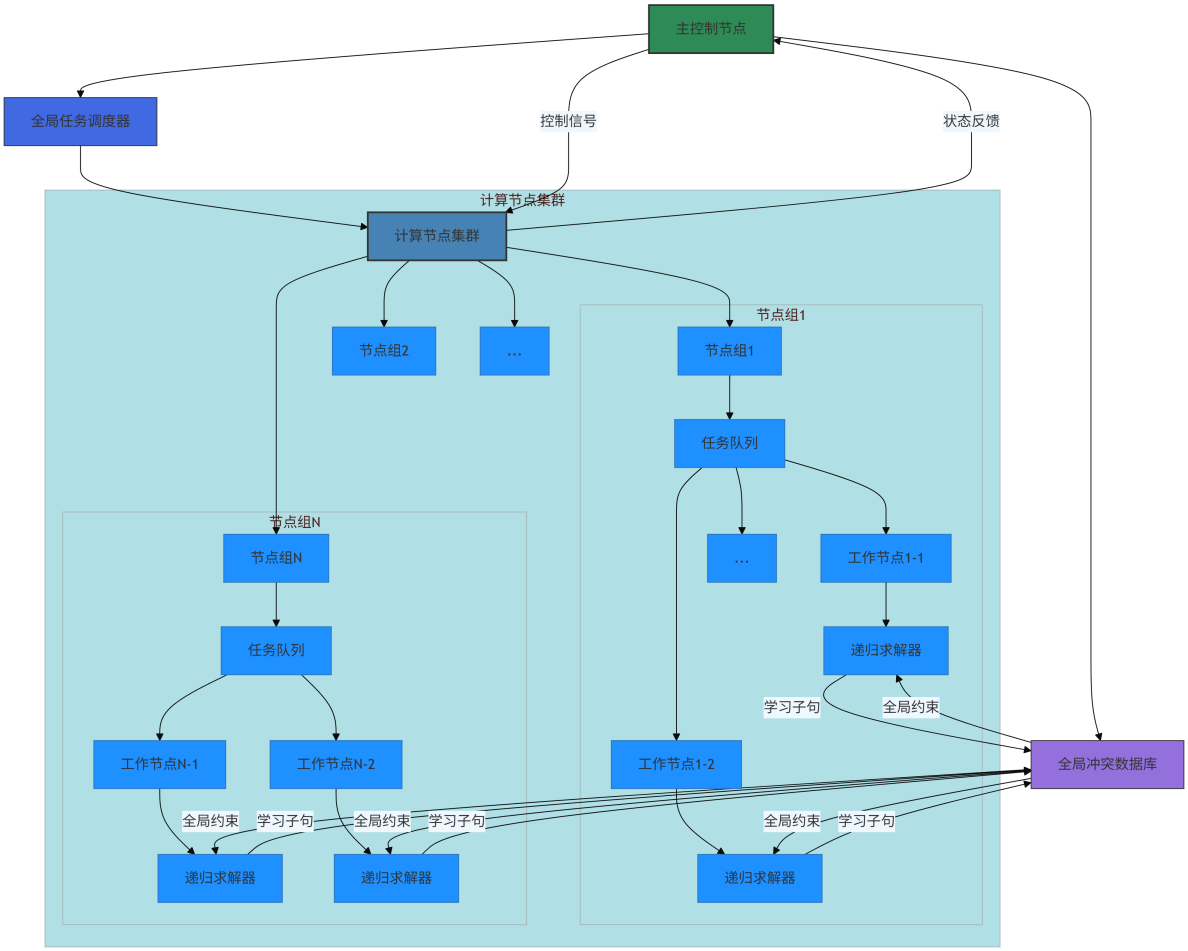


下面可以直观的感受递归的过程：




这么多的并行度适合应用在超算求解之中：

下面是对SAT超算求解系统的设计：




超算一般都拥有几万个节点，很适合这种超大规模的并行度计算。