SAT 求解導論

完整教材 (依據 5CCS2FC2 Foundations of Computing II 第五週內容編寫)

受限 SAT、2-SAT 與蘊涵圖、DPLL 與貪婪局部搜尋

整理自 Dr. Christopher Hampson(King's College London) 之投影片
並參考補充文獻擴充編寫

2026 年 6 月

本週學習目標

1. 理解 **SAT 的重要性**: 第一個 NP-complete 問題、所有 NP 問題的「通用語言」。
2. 掌握**受限版本** N**-Sat** 與兩條多項式歸約鏈, 特別是 SAT $\leq_p$ 3-SAT 的子句拆分技巧。
3. 理解**為何 2-Sat 屬於 P**: 蘊涵圖 + 強連通元件, 並掌握其充要條件與正確性證明。
4. 認識另一個容易的變體:**Horn 公式**。
5. 掌握實務 SAT 求解的核心: **樸素分支**、**DPLL**(單位傳播、純文字消去) 與**貪婪局部搜尋 (GSAT)**。

Contents

1 為什麼 SAT 如此重要?

布林可滿足性問題 SAT 是整個計算複雜度理論的核心。回顧第二、三週:

- SAT 是**第一個**被證明為 NP-complete 的問題 (Cook–Levin 定理)。
- 所有 **NP** 問題都能在多項式時間內歸約到 SAT。
- 因此, 若能找到 SAT 的多項式時間演算法, 就證明了 **P** = **NP**(百萬美元問題)。

SAT 作為「規範語言」

邏輯是描述現實問題的**規範語言 (specification language)**: 排程、電路驗證、人工智慧規劃、軟體測試等, 都能直接編碼成 SAT。把問題歸約到 SAT 通常很直接。即使 **P** $\neq$ **NP**, 我們仍希望盡快回答這些實際問題——於是有了 **SAT 求解器 (SAT Solvers)**。現代求解器能處理含**數百萬個文字**的公式, 廣泛應用於工業界。

術語回顧 一個 CNF(合取範式) 公式是「子句的 AND」, 每個**子句 (clause)** 是「文字的 OR」, 每個**文字 (literal)** 是某變數 P 或其否定 $\neg P$。例如

$$\underbrace{(P \vee \neg Q \vee R)}_{\text{子句}} \wedge (\neg P \vee S) \wedge (\neg R \vee \neg S).$$

2 受限版本的 SAT:N-Sat

定義 2.1 (N-SAT)**.** 布林可滿足性問題 N-SAT:

輸入 一個 CNF 公式 F, 其中每個子句至多 N 個文字。

輸出 **True** 若且唯若 F 可滿足。

(有些定義要求恰好 N 個文字, 以下的歸約鏈對兩種定義皆成立。)

2.1 容易的方向 (短子句歸約到長子句)

定理 2.2. 存在以下多項式歸約鏈*:*

$$2\text{-}SAT \leq_p 3\text{-}SAT \leq_p 4\text{-}SAT \leq_p \cdots \leq_p k\text{-}SAT \leq_p \cdots \leq_p SAT.$$

Proof. 這方向很容易: 任何「每子句至多 k 個文字」的 CNF 公式, 當然也是「每子句至多 $(k+1)$ 個文字」的公式。故恆等映射就是一個合法歸約。 □

2.2 令人驚訝的方向 (SAT 歸約到 3-SAT)

定理 2.3. 也存在反向的歸約鏈*:*

$$SAT \leq_p \cdots \leq_p k\text{-}SAT \leq_p \cdots \leq_p 5\text{-}SAT \leq_p 4\text{-}SAT \leq_p 3\text{-}SAT.$$

只需證明關鍵一步 SAT $\leq_p$ 3-SAT(把任意長子句壓到至多 3 個文字); 其餘 $(k{+}1)$-SAT $\leq_p$ k-SAT 同理。核心技巧是**子句拆分 + 引入新變數**。

子句拆分: 把長子句切短

步驟 1. 任取一個 CNF 公式:

$$(P \vee \neg Q \vee R \vee S) \wedge (Q \vee \neg R \vee \neg T).$$

步驟 2. 找出含超過 3 個文字的子句 (此處第一個有 4 個)。
步驟 3. 從中間把它**切成兩半**:$(P \vee \neg Q \ \mid \ R \vee S)$。
步驟 4. 引入一個**全新變數** X 來「搭橋」:

$$(P \vee \neg Q \vee X) \wedge (\neg X \vee R \vee S) \wedge (Q \vee \neg R \vee \neg T).$$

步驟 5. 重複, 直到所有子句至多 3 個文字。

備註 2.4 (為什麼保持可滿足性). 新變數 X 的作用是「二選一開關」:

- 若原子句靠前半 $(P \vee \neg Q)$ 被滿足, 令 $X = \text{False}$, 則 $(\neg X \vee R \vee S)$ 自動為真;
- 若靠後半 $(R \vee S)$ 被滿足, 令 $X = \text{True}$, 則 $(P \vee \neg Q \vee X)$ 自動為真。

反之, 若兩半都不滿足, 則無論 X 取何值, 兩個新子句必有一個為假。因此

$$\text{新公式可滿足} \iff \text{原公式可滿足}.$$

一個 k 文字子句最多被拆成 $k-2$ 個 3 文字子句, 引入 $k-3$ 個新變數, 公式只增大線性倍, 故為多項式歸約。

2.3 懸崖: 能再歸約到 2-SAT 嗎?

歸約鏈走到 3-SAT 之後, 自然會問: 能否再往下 3-SAT $\leq_p$ 2-SAT?

關鍵分界線

不能 (除非 P = NP)。3-SAT 仍是 NP-complete, 但 2-SAT 卻在 **P** 中! 這是複雜度的一道「懸崖」:

$$\boxed{\text{2-SAT} \in \mathbf{P} \quad \text{但} \quad \text{3-SAT 是 NP-complete}}$$

從「每子句 2 個文字」到「3 個文字」, 問題難度發生質變。下一節說明為何 2-SAT 如此特別。

3 2-Sat 的多項式時間解法

定理 3.1 (Aspvall–Plass–Tarjan, 1979)**.** 2-*SAT* 可在多項式 *(實則線性)* 時間內求解。

證明的核心想法: 把 2-SAT 轉化成**強連通元件 (SCC)** 問題——正是第四週學過的工具!

3.1 步驟一: 把子句改寫成蘊涵

關鍵等價式: 子句 $(a \vee b)$ 邏輯上等於兩條蘊涵

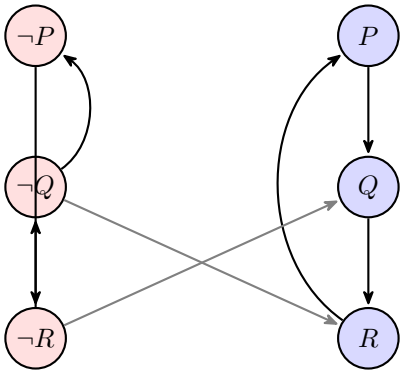
$$(a \vee b) \ \equiv \ (\neg a \to b) \wedge (\neg b \to a).$$

直覺:「a 或 b 至少一真」$\Leftrightarrow$「若 a 假則 b 必真, 且若 b 假則 a 必真」。以投影片的例子:

子句	等價的兩條蘊涵
$(\neg P \vee Q)$	$(P \to Q) \ \wedge \ (\neg Q \to \neg P)$
$(\neg Q \vee R)$	$(Q \to R) \ \wedge \ (\neg R \to \neg Q)$
$(P \vee \neg R)$	$(\neg P \to \neg R) \ \wedge \ (R \to P)$
$(R \vee Q)$	$(\neg R \to Q) \ \wedge \ (\neg Q \to R)$

3.2 步驟二: 建立蘊涵圖

蘊涵圖 (implication graph) 的頂點是所有文字 $(P, \neg P, Q, \neg Q, \dots)$, 每條蘊涵 $\ell_1 \to \ell_2$ 對應一條有向邊。



3.3 步驟三、四: 計算 SCC 並檢查矛盾

用第四週的 Kosaraju 演算法計算 SCC。上圖的強連通元件為

$$\{P, Q, R\} \qquad \text{與} \qquad \{\neg P, \neg Q, \neg R\}.$$

接著**檢查: 是否有任一 SCC 同時含一個文字與其否定?** 此例中 $\{P, Q, R\}$ 與 $\{\neg P, \neg Q, \neg R\}$ 皆不含互補文字, 故公式**可滿足**。✓

定理 3.2 (2-SAT 的充要條件)**.** 一個 *2-CNF* 公式**可滿足** $\iff$ 蘊涵圖中沒有任何變數 x 使得 x 與 $\neg x$ 落在**同一個 *SCC***。

證明. ($\Rightarrow$, 反面) 若 x 與 $\neg x$ 同屬一個 SCC, 則圖中既有路徑 $x \rightsquigarrow \neg x$、又有 $\neg x \rightsquigarrow x$。沿著蘊涵路徑,$x$ 為真會強迫 $\neg x$ 為真 (反之亦然), 這對任何指派都導致矛盾。故公式不可滿足。

($\Leftarrow$) 設無任何變數與其否定同 SCC。把各 SCC 依**拓撲序**排列 (回憶第四週: 凝聚圖必為 DAG)。對每個變數 x, 令

$$x := \text{True} \iff \text{comp}(x) \text{ 排在 } \text{comp}(\neg x) \text{ 之後 (拓撲序較大)}.$$

可證此指派滿足所有蘊涵邊 (不存在「真 $\to$ 假」的邊), 故滿足原公式。此外,x 與 $\neg x$ 所在元件在凝聚圖中恰好對稱, 保證指派一致。 □

推論 3.3. 由於建圖、求 *SCC*、檢查互補皆為 $O(|V| + |E|)$,*2-SAT* 可在線性時間內求解並同時給出一組滿足指派。

4 另一個容易的變體:Horn 公式

定義 4.1 ((定言)Horn 子句)**.** 一個 **Horn 子句**是至多含一個正文字的子句。**定言 (definite)Horn 子句**則恰含一個正文字, 可寫成「規則」形式:

$$(\neg P_1 \vee \cdots \vee \neg P_k \vee H) \;\equiv\; (P_1 \wedge \cdots \wedge P_k \to H).$$

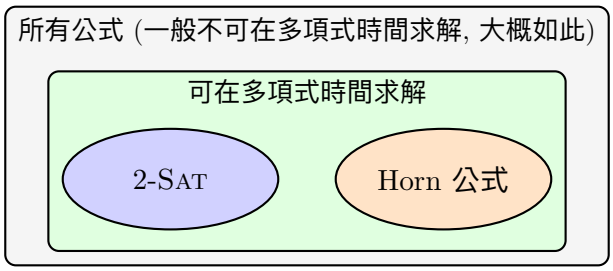
即「若前提 $P_1, \ldots, P_k$ 皆真, 則結論 (頭)H 為真」。

例 4.2.

$$P, \quad (P \to Q), \quad (P \wedge Q \to R), \quad (Q \to S), \quad (S \wedge R \to T).$$

從事實 P 出發, 反覆套用規則 (**單位傳播**):$P \Rightarrow Q \Rightarrow R$(由 $P \wedge Q$)、S(由 Q)、T(由 $S \wedge R$)。

Horn-SAT 在 P 中: 反覆做單位傳播 (每次把被強迫為真的頭加入指派), 直到沒有新的可推導, 或推出矛盾。此過程為線性時間。Horn 公式是邏輯程式 (如 Prolog) 與規則系統的理論基礎。



5 樸素分支演算法

對一般的 SAT, 沒有已知的多項式演算法, 但我們仍需實用的求解器。最基本的是**分支 (branching) 分治演算法**: 對某個變數 P, 分別嘗試 $P = \text{False}$ 與 $P = \text{True}$, 遞迴求解。

Algorithm 1 BRANCH-SAT(C, U)

輸入: 子句集 C、部分指派 $U \subseteq \{C$ 中的文字$\}$

1: **if** U 含矛盾 **then return False**
2: **if** $C = \varnothing$ **then return True**
3: 盡量化簡 C 中的子句
4: 選一個出現在 C 中的變數 P
5: **if** BRANCH-SAT$(C, U \cup \{\neg P\}) =$ **True then return True**
6: **if** BRANCH-SAT$(C, U \cup \{P\}) =$ **True then return True**
7: **return False**

每分支一次, 變數少一個, 故子問題與原問題自相似但規模減一。最壞情況需探索 2^n 個葉節點 (指數時間), 但搭配下一節的化簡規則, 實務上能剪掉大量分支。

6 DPLL 演算法

DPLL(Davis–Putnam–Logemann–Loveland) 演算法於 1962 年提出 (承襲 Davis 與 Putnam 1960 年的工作), 是現代 SAT 求解器的鼻祖。它在樸素分支的骨架上, 加入兩個威力強大的化簡規則:

純文字消去與單位傳播。

6.1 純文字消去 (Pure Literal Elimination)

定義 6.1. 若某文字在所有子句中只以**單一極性**出現 (只出現正、或只出現負), 稱為**純文字 (pure literal)**。

例 6.2.

$$(P \vee Q \vee \neg S),\ (P \vee \neg Q),\ (Q \vee R),\ (Q \vee \neg R \vee \neg S).$$

P 只以正極性出現、S 只以負極性出現, 故 $P, \neg S$ 為純文字。可無風險地令

$$P := \text{True}, \qquad S := \text{False},$$

並**刪去**所有含這些純文字的子句 (它們已被滿足)。

直覺

純文字消去指派的是**「應該」(SHOULD)** 被指派的文字: 把純文字設為使它為真, 只可能讓更多子句被滿足、絕不會造成矛盾, 所以放心指派。

6.2 單位傳播 (Unit Propagation)

定義 6.3. 只含單一文字的子句稱為**單位子句 (unit clause)**。

對單位子句, 我們別無選擇: 那個文字**必須**為真。例如有單位子句 $\neg Q$ 與 R, 則強制 $Q := \text{False}$、$R := \text{True}$。接著:

- **刪去**所有含該文字的子句 (已滿足);
- 從其餘子句中**刪掉**該文字的否定 (永遠為假, 無貢獻)。

這些指派會連鎖傳播, 往往觸發更多單位子句, 形成「骨牌效應」。

單位傳播的連鎖過程

起始 (含單位子句 $\neg Q$、R):

$$\neg Q,\ (P \vee Q),\ (\neg P \vee \neg R \vee S),\ (\neg P \vee Q \vee \neg S),\ (\neg P \vee \neg Q),\ R,\ (Q \vee R \vee S).$$

由 $\neg Q(Q{:=}\text{F})$ 與 $R(R{:=}\text{T})$ 化簡:$(P \vee Q) \to P$(單位!),$(\neg P \vee \neg R \vee S) \to (\neg P \vee S)$,$(\neg P \vee Q \vee \neg S) \to (\neg P \vee \neg S)$.....。再由新單位子句 $P{:=}\text{T}$ 繼續傳播, 得 S 與 $\neg S$——**矛盾!** 此分支回溯。

直覺

單位傳播指派的是**「必須」(MUST)** 被指派的文字: 沒有選擇餘地。(對照: 純文字消去是「應該」, 單位傳播是「必須」。)

6.3 完整的 DPLL 演算法

Algorithm 2 DPLL(C, U)

輸入: 子句集 C、部分指派 U

if U 含矛盾 **then return False**
if $C = \varnothing$ **then return True**
盡量化簡 C 中的子句
對 C 施行純文字消去
對 C 施行單位傳播
選一個出現在 C 中的變數 P
if DPLL$(C,\, U \cup \{\neg P\}) =$ **True then return True**
if DPLL$(C,\, U \cup \{P\}) =$ **True then return True**
return False

DPLL 是**完備 (complete)** 的: 一定能正確判定可滿足與否。化簡規則大幅縮小搜尋樹, 但最壞情況仍是指數時間 (畢竟 SAT 是 NP-complete)。

從 DPLL 到現代 CDCL 求解器

1990 年代後, **衝突驅動子句學習 (Conflict-Driven Clause Learning, CDCL)** 在 DPLL 骨架上加入三大關鍵技術:

- **子句學習**: 遇到衝突時「學會」其根本原因, 加入一條新子句以避免重蹈覆轍;
- **非時序回溯 (backjumping)**: 一次跳回多層, 而非逐層回溯;
- **高效資料結構**: 如「兩監視文字 (two watched literals)」加速單位傳播, 以及 VSIDS 分支啟發式、重啟策略等。

現今最先進的 SAT 求解器幾乎全部基於 CDCL, 能處理含數百萬變數的工業級實例。

7 貪婪局部搜尋 (GSAT)

另一條路線是**隨機局部搜尋**: 不做系統化的分支, 而是從一個隨機指派出發, 反覆「翻轉」單一變數以爬向更好的解。

Algorithm 3 Greedy-SAT(C)

隨機選一組變數指派 V
while C 中仍有未滿足的子句 **do**
    計算目前被滿足的子句數 (「分數」)
    for 每個變數 P **do**
        計算「翻轉 P」後被滿足的子句數
    if 沒有任何翻轉能改善 **then return False**
    把 V 更新為改善最多的那次翻轉
return True

7.1 範例: 爬山過程與局部最大值

考慮 7 個子句 $(P, Q, R$ 三變數):

$$(P \vee Q),\ (P \vee R),\ (P \vee \neg Q \vee R),\ (\neg P \vee \neg Q),$$
$$(\neg P \vee Q),\ (\neg P \vee \neg R),\ (P \vee Q \vee \neg R).$$

列出全部 8 種指派的「分數」(被滿足的子句數, 滿分 7):

P	Q	R	指派	分數
F	F	F	(F,F,F)	5
T	F	F	(T,F,F)	6
F	T	F	(F,T,F)	5
F	F	T	(F,F,T)	5
T	T	F	(T,T,F)	6
T	F	T	(T,F,T)	5
F	**T**	**T**	**(F,T,T)**	**7** ✓
T	T	T	(T,T,T)	5

唯一的滿足指派是 $(P, Q, R) = (\mathrm{F}, \mathrm{T}, \mathrm{T})$, 分數 7。

陷阱: 貪婪會卡在局部最大值

若從 $(\mathrm{F}, \mathrm{F}, \mathrm{F})$(分數 5) 出發, 單一翻轉的三個鄰居分數為:(T,F,F) $\to$ 6、(F,T,F) $\to$ 5、(F,F,T) $\to$ 5。貪婪選最佳的 (T,F,F) $=$ 6。但 (T,F,F) 的鄰居 (F,F,F)5、(T,T,F)6、(T,F,T)5 都沒有改善——演算法卡在**局部最大值** 6, 回傳 **False**, 卻錯過了真正存在的解 $(\mathrm{F},\mathrm{T},\mathrm{T}) = 7$!

定理 7.1. *GREEDY-SAT* 的每一輪 *(*每次爬升*)* 在多項式時間內完成。

Proof.
- 隨機選指派、計算分數, 皆為線性時間。
- 每一輪至多評估 n 個「翻轉單一變數」的指派, 共多項式步數。
- 由於分數從不下降且上界為子句總數, 至多迭代多項式次。

□

備註 7.2 (完備性的代價). GSAT 雖然每次執行都是多項式時間, 但它**不完備**: 可能卡在局部最大值而回傳 False, 即使公式其實可滿足。實務上以**隨機重啟 (random restarts)** 與**隨機 walk 步 (如 WalkSAT)** 緩解——容許偶爾走「非最佳」甚至「變差」的一步以跳出局部最大值。這類隨機求解器對某些大型實例非常有效, 但無法證明不可滿足。

	DPLL / CDCL	**GSAT / WalkSAT**
策略	系統化分支 + 化簡	隨機局部搜尋 (爬山)
完備性	完備 (可判定 SAT 與 UNSAT)	不完備 (只能找解, 無法證 UNSAT)
最壞時間	指數	每輪多項式, 但可能找不到

8 本週重點整理

主題	重點
N-SAT 歸約鏈	2-SAT $\le_p$ 3-SAT $\le_p \cdots \le_p$ SAT(易); 反向需子句拆分 SAT $\le_p$ 3-SAT
複雜度懸崖	2-SAT $\in \mathbf{P}$, 但 3-SAT 是 NP-complete
2-SAT 解法	蘊涵圖 + SCC; 可滿足 $\iff$ 無變數與其否定同 SCC(線性時間)
Horn 公式	每子句至多一個正文字; 單位傳播即可線性求解
Branch-SAT	對變數分支的分治法, 最壞 2^n
DPLL	分支 + 純文字消去 (SHOULD) + 單位傳播 (MUST); 完備
CDCL	DPLL + 子句學習 + backjumping; 現代工業級求解器
GSAT	隨機爬山, 每輪多項式但**不完備** (易卡局部最大值)

9 練習題 (附解答)

練習 1. 把子句 $(\neg A \vee B)$ 改寫為兩條蘊涵, 並說出它在蘊涵圖中對應哪兩條邊。

解答

$(\neg A \vee B) \equiv (A \to B) \wedge (\neg B \to \neg A)$。對應邊:$A \to B$ 與 $\neg B \to \neg A$。

練習 2. 用蘊涵圖判斷 $(x \vee y) \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg z)$ 是否可滿足。

解答

由後兩子句:$y \to z$ 且 $y \to \neg z$, 故 y 為真會強迫 z 與 $\neg z$ 同真, 矛盾; 由前兩子句 $\neg x \to y$、$x \to y$, 故 y 必為真。於是 y 與 $\neg y$ 互相可達、同屬一 SCC, **不可滿足**。(亦可直接看出:y 被強迫為真卻又導致 z 矛盾。)

練習 3. 把長子句 $(A \vee B \vee C \vee D \vee E)$ 用子句拆分法化為至多 3 文字的子句。

解答

引入新變數 X_1, X_2:

$$(A \vee B \vee X_1) \wedge (\neg X_1 \vee C \vee X_2) \wedge (\neg X_2 \vee D \vee E).$$

5 文字子句拆成 $5 - 2 = 3$ 個 3 文字子句, 新增 2 個變數。

練習 4. 在公式 $(P \vee \neg Q)$, $(P \vee R)$, $(\neg R)$ 中, 找出純文字與單位子句, 並做一步化簡。

解答

$\neg R$ 是**單位子句** $\Rightarrow R :=$ False, 化簡 $(P \vee R) \to P$(又成單位子句)$\Rightarrow P :=$ True, 兩個含 P 的子句被滿足。P 只以正極性出現, 亦是**純文字**。剩餘無子句, 公式可滿足 (P=T, R=F,Q 任意)。

練習 5 (思考題)**.** 為什麼 GSAT 能在多項式時間內跑完每一輪, 卻無法用來證明 $\mathbf{P} = \mathbf{NP}$?

解答

GSAT 不完備: 它可能卡在局部最大值而回傳 False, 即使公式可滿足; 它也無法證明「不可滿足」。要證 $\mathbf{P} = \mathbf{NP}$ 需要一個完備且保證多項式時間的 SAT 演算法。GSAT 只滿足「多項式時間」卻犧牲了「完備性」, 故無助於解決百萬美元問題。

參考資料

1. C. Hampson, *5CCS2FC2 Foundations of Computing II, Week 5* 投影片 (2sat / dpll / greedy),King's College London.
2. B. Aspvall, M. F. Plass, R. E. Tarjan, "A linear-time algorithm for testing the truth of certain quantified Boolean formulas," *Information Processing Letters*, 8(3):121–123, 1979.
3. M. Davis, G. Logemann, D. Loveland, "A machine program for theorem-proving," *Comm. ACM*, 5(7):394–397, 1962.
4. A. Biere, M. Heule, H. van Maaren, T. Walsh (eds.), *Handbook of Satisfiability*, IOS Press. (DPLL、CDCL、局部搜尋章節)
5. B. Selman, H. Levesque, D. Mitchell, "A new method for solving hard satisfiability problems (GSAT)," *AAAI*, 1992.
6. Wikipedia: *2-satisfiability*; *DPLL algorithm*; *Conflict-driven clause learning*; *Horn-satisfiability*. CP-Algorithms: *2-SAT*.