

IL PROBLEMA SAT ED IL SUO IMPATTO

Agostino Dovier

Dept of Mathematics, Computer Science, and Physics
University of Udine
Udine (Italy)

PERUGIA, 2018

SOMMARIO

Vedremo

- il problema SAT
- il suo ruolo nel problema (del millennio) aperto **P vs NP**
- gli approcci (esponenziali nel caso peggiore) alla sua risoluzione
- come si possano sfruttare positivamente i risultati di tali approcci per risolvere altri problemi
- un linguaggio per modellare esattamente i problemi in NP

SAT

- Dato un insieme di variabili Booleane $\mathcal{X} = \{X_1, \dots, X_n\}$
- e una formula proposizionale in CNF costruita su \mathcal{X} :

$$\Phi = (\ell_1^1 \vee \dots \vee \ell_{n_1}^1) \wedge \dots \wedge (\ell_1^k \vee \dots \vee \ell_{n_k}^k)$$

dove ogni ℓ_j^i è una variabile X_p o la sua negazione $\neg X_p$

- Il problema **SAT** è quello di **stabilire l'esistenza** di un assegnamento di valori **vero/falso** (**1/0**) alle variabili in grado di rendere vera la formula Φ
- Ricordiamo che:
 $\neg 0 = 1, \neg 1 = 0$
 $0 \vee 0 = 0, 0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$
 $0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0, 1 \wedge 1 = 1$

SAT

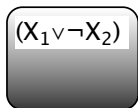
ESEMPIO

$$(X_1 \vee \neg X_2) \quad (\neg X_1 \vee X_2 \vee \neg X_3) \quad (\neg X_1 \vee \neg X_2 \vee X_3) \quad (\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono “tante” $(2^{|\mathcal{X}|})$.

SAT

ESEMPIO



$$(\neg X_1 \vee X_2 \vee \neg X_3)$$

$$(\neg X_1 \vee \neg X_2 \vee X_3)$$

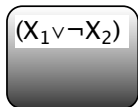
$$(\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

$$X_1=1$$

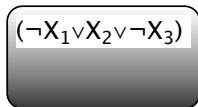
1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono “tante” $(2^{|\mathcal{X}|})$.

SAT

ESEMPIO



$$X_1=1$$



$$X_2=1$$

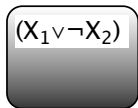
$$(\neg X_1 \vee \neg X_2 \vee X_3)$$

$$(\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

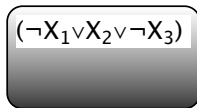
1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono “tante” $(2^{|\mathcal{X}|})$.

SAT

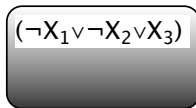
ESEMPIO



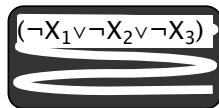
$$X_1=1$$



$$X_2=1$$



$$X_3=1$$



1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono “tante” $(2^{|\mathcal{X}|})$.

SAT

ESEMPIO

$$(X_1 \vee \neg X_2)$$

$$X_1 = 1$$

$$(\neg X_1 \vee X_2 \vee \neg X_3)$$

$$X_2 = 1$$

$$(\neg X_1 \vee \neg X_2 \vee X_3)$$

$$X_3 = 0$$

$$(\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono “tante” $(2^{|\mathcal{X}|})$.

SAT

ESEMPIO

$$(X_1 \vee \neg X_2)$$

$$(\neg X_1 \vee X_2 \vee \neg X_3)$$

$$(\neg X_1 \vee \neg X_2 \vee X_3)$$

$$(\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

$$X_1=1$$

$$X_3=0$$

$$X_2=0$$

1. Data una possibile soluzione, verificarla è facile.
2. Le possibili soluzioni sono “tante” ($2^{|X|}$).

SAT

LA CNF È IMPORTANTE?

$$\frac{(((X_1 \wedge X_2) \vee (X_3 \wedge \neg X_4)) \wedge (X_6 \vee (X_7 \wedge \neg X_8))) \vee X_9}{(*)}$$

$$((A \vee B) \wedge (X_6 \vee C)) \vee X_9 \quad (*)$$

$$A \leftrightarrow (X_1 \wedge X_2)$$

$$B \leftrightarrow (X_3 \wedge \neg X_4)$$

$$C \leftrightarrow (X_7 \wedge \neg X_8)$$

$$\frac{(D \wedge E) \vee X_9}{(*)}$$

$$D \leftrightarrow (A \vee B)$$

$$E \leftrightarrow (X_6 \vee C)$$

$$F \vee X_9$$

$$F \leftrightarrow (D \wedge E)$$

SAT

LA CNF È IMPORTANTE?

$$\frac{(((X_1 \wedge X_2) \vee (X_3 \wedge \neg X_4)) \wedge (X_6 \vee (X_7 \wedge \neg X_8))) \vee X_9}{((A \vee B) \wedge (X_6 \vee C)) \vee X_9} \quad (*)$$

$$A \leftrightarrow (X_1 \wedge X_2)$$

$$B \leftrightarrow (X_3 \wedge \neg X_4)$$

$$C \leftrightarrow (X_7 \wedge \neg X_8)$$

$$\frac{(D \wedge E) \vee X_9}{D \leftrightarrow (A \vee B)} \quad (*)$$

$$E \leftrightarrow (X_6 \vee C)$$

$$F \vee X_9$$

$$F \leftrightarrow (D \wedge E)$$

SAT

LA CNF È IMPORTANTE?

$$\begin{array}{rcl}
 (((X_1 \wedge X_2) \vee (X_3 \wedge \neg X_4)) \wedge (X_6 \vee (X_7 \wedge \neg X_8))) \vee X_9 & (*) \\
 \hline
 ((A \vee B) \wedge (X_6 \vee C)) \vee X_9 & (*) \\
 A \leftrightarrow (X_1 \wedge X_2) \\
 B \leftrightarrow (X_3 \wedge \neg X_4) \\
 C \leftrightarrow (X_7 \wedge \neg X_8) \\
 \hline
 (D \wedge E) \vee X_9 & (*) \\
 D \leftrightarrow (A \vee B) \\
 E \leftrightarrow (X_6 \vee C) \\
 \hline
 F \vee X_9 \\
 F \leftrightarrow (D \wedge E)
 \end{array}$$

SAT

LA CNF È IMPORTANTE?

$$\begin{array}{rcl}
 (((X_1 \wedge X_2) \vee (X_3 \wedge \neg X_4)) \wedge (X_6 \vee (X_7 \wedge \neg X_8))) \vee X_9 & (*) \\
 \hline
 ((A \vee B) \wedge (X_6 \vee C)) \vee X_9 & (*) \\
 A \leftrightarrow (X_1 \wedge X_2) & \\
 B \leftrightarrow (X_3 \wedge \neg X_4) & \\
 C \leftrightarrow (X_7 \wedge \neg X_8) & \\
 \hline
 (D \wedge E) \vee X_9 & (*) \\
 D \leftrightarrow (A \vee B) & \\
 E \leftrightarrow (X_6 \vee C) & \\
 \hline
 F \vee X_9 & \\
 F \leftrightarrow (D \wedge E) &
 \end{array}$$

SAT

LA CNF È IMPORTANTE?

Ove

$$U \leftrightarrow (V \wedge Z) \quad =$$

$$U \rightarrow (V \wedge Z) \wedge (V \wedge Z) \rightarrow U \quad =$$

$$(\neg U \vee V) \wedge (\neg U \vee Z) \wedge (\neg V \vee \neg Z \vee U)$$

e

$$U \leftrightarrow (V \vee Z) \quad =$$

$$U \rightarrow (V \vee Z) \wedge (V \vee Z) \rightarrow U \quad =$$

$$(\neg U \vee V \vee Z) \wedge (\neg V \vee U) \wedge (\neg Z \vee U)$$

No, non sarebbe importante. E' comodo ragionare su una forma normale. Data una Φ proposizionale la possiamo preprocessare con le idee viste sopra.

SAT

LA CNF È IMPORTANTE?

Ove

$$U \leftrightarrow (V \wedge Z) \quad =$$

$$U \rightarrow (V \wedge Z) \wedge (V \wedge Z) \rightarrow U \quad =$$

$$(\neg U \vee V) \wedge (\neg U \vee Z) \wedge (\neg V \vee \neg Z \vee U)$$

e

$$U \leftrightarrow (V \vee Z) \quad =$$

$$U \rightarrow (V \vee Z) \wedge (V \vee Z) \rightarrow U \quad =$$

$$(\neg U \vee V \vee Z) \wedge (\neg V \vee U) \wedge (\neg Z \vee U)$$

No, non sarebbe importante. E' comodo ragionare su una forma normale. Data una Φ proposizionale la possiamo preprocessare con le idee viste sopra.

SAT

E LA DNF?

$$(X_1 \wedge \neg X_2 \wedge X_3) \vee (\neg X_1 \wedge X_2) \vee \dots$$

Qui il problema sarebbe banale.

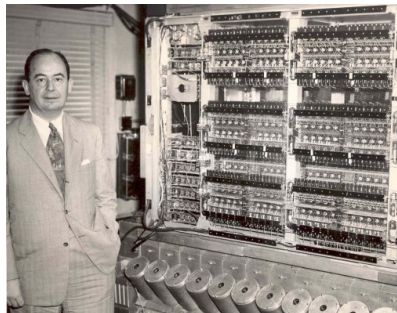
La DNF è una descrizione delle soluzioni!!!

LA LETTERA DI GÖDEL A VON NEUMANN

[HTTPS://RJLIPTON.WORDPRESS.COM/THE-GDEL-LETTER/](https://rjlipton.wordpress.com/the-gdel-letter/)



(1906–1978)



(1903–1957)

LA LETTERA DI GÖDEL A VON NEUMANN

[HTTPS://RJLIPTON.WORDPRESS.COM/THE-GDEL-LETTER/](https://rjlipton.wordpress.com/the-gdel-letter/)

Princeton, 20 March 1956

Lieber Herr v. Neumann:

With the greatest sorrow I have learned of your illness. [...] I would like to allow myself to write you about a mathematical problem, of which your opinion would very much interest me: One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n , allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\psi(F, n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F, n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. [...] If there really were a machine with $\varphi(n) \sim kn$ (or even $\sim kn^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. [...] Now it seems to me, however, to be completely within the realm of possibility that $\varphi(n)$ grows that slowly. [...]

SAT E GÖDEL

$$\Phi = (X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2)$$

La possiamo pensare “alla Gödel” come

$$F = \exists X_1 \exists X_2 \exists X_3 ((p(X_1) \vee \neg p(X_2) \vee p(X_3)) \wedge (\neg p(X_1) \vee p(X_2)))$$

Dunque Gödel era **ottimista** sull'esistenza di un algoritmo polinomiale (addirittura lineare o quadratico) per la risoluzione del problema SAT.

Purtroppo non sappiamo cosa ne pensasse von Neumann (che morì di Cancro poco dopo).

60 anni dopo non sappiamo ancora se Gödel avesse ragione. Nè se avesse torto.

LA CLASSE P

Un problema/linguaggio $L \subseteq \Sigma^*$ appartiene alla classe **P** se c'è un algoritmo A ed un polinomio p t.c. per ogni $x \in \Sigma^*$:

$$x \in L \leftrightarrow A(x) = \text{yes}$$

e ogni computazione di A termina in al più $p(|x|)$ passi.

A , in breve, si dice **algoritmo polinomiale**. Dunque ad esempio A **opera** in tempo $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, etc. (la notazione “ O ” si usa per trascurare costanti moltiplicative e termini di grado inferiore)

Esempio $L = \{x \in \Sigma^* : x \text{ è palindroma}\}$.

LA CLASSE NP

Un problema/linguaggio $L \subseteq \Sigma^*$ appartiene alla classe **NP** se c'è un algoritmo *polinomiale* A ed un polinomio q t.c. per ogni $x \in \Sigma^*$:

$$x \in L \leftrightarrow \exists c \in \Sigma^{q(|x|)} (A(x, c) = \text{yes})$$

c è detto **certificato (guess)** per x , di dimensione limitata da $q(|x|)$ che ci permette di verificare (**verify**) o di refutare che $x \in L$ in tempo polinomiale.

Esempio: SAT. Il certificato è l'assegnamento $X_1/0, X_2/1, \dots, X_n/1$ delle variabili.

Cercare UN certificato che verifichi è la parte difficile. Tipicamente dobbiamo esplorare uno spazio di ricerca di dimensioni esponenziali rispetto a $|x|$.

LA CLASSE NP

Un problema/linguaggio $L \subseteq \Sigma^*$ appartiene alla classe **NP** se c'è un algoritmo *polinomiale* A ed un polinomio q t.c. per ogni $x \in \Sigma^*$:

$$x \in L \leftrightarrow \exists c \in \Sigma^{q(|x|)} (A(x, c) = \text{yes})$$

c è detto **certificato (guess)** per x , di dimensione limitata da $q(|x|)$ che ci permette di verificare (**verify**) o di refutare che $x \in L$ in tempo polinomiale.

Esempio: SAT. Il certificato è l'assegnamento $X_1/0, X_2/1, \dots, X_n/1$ delle variabili.

Cercare UN certificato che verifichi è la parte difficile. Tipicamente dobbiamo esplorare uno **spazio di ricerca** di dimensioni esponenziali rispetto a $|x|$.

RIDUZIONI

Dati due problemi/linguaggi A e B , A si riduce a B (in breve $A \leq B$) se esiste una funzione f (di riduzione che **trasforma una istanza di un problema di A in una istanza di problema di B**) calcolabile da un algoritmo “efficiente” (tecnicamente in **spazio logaritmico**, ma per questa presentazione pensate pure ad un algoritmo polinomiale) tale che

$$\forall x (x \in A \leftrightarrow f(x) \in B)$$

E' evidente che se $B \in P$ e $A \leq B$ allora $A \in P$.
Similmente, se $B \in NP$ e $A \leq B$ allora $A \in NP$.

Un problema/linguaggio $L \in P$ è P completo se ogni linguaggio in P si può ridurre a lui. Un problema/linguaggio $L \in NP$ è NP completo se ogni linguaggio in NP si può ridurre a lui.

RIDUZIONI

Dati due problemi/linguaggi A e B , A si riduce a B (in breve $A \leq B$) se esiste una funzione f (di riduzione che **trasforma una istanza di un problema di A in una istanza di problema di B**) calcolabile da un algoritmo “efficiente” (tecnicamente in **spazio logaritmico**, ma per questa presentazione pensate pure ad un algoritmo polinomiale) tale che

$$\forall x (x \in A \leftrightarrow f(x) \in B)$$

E' evidente che se $B \in P$ e $A \leq B$ allora $A \in P$.
Similmente, se $B \in NP$ e $A \leq B$ allora $A \in NP$.

Un problema/linguaggio $L \in P$ è P completo se ogni linguaggio in P si può ridurre a lui. Un problema/linguaggio $L \in NP$ è NP completo se ogni linguaggio in NP si può ridurre a lui.

RIDUZIONI

Dati due problemi/linguaggi A e B , A si riduce a B (in breve $A \leq B$) se esiste una funzione f (di riduzione che **trasforma una istanza di un problema di A in una istanza di problema di B**) calcolabile da un algoritmo “efficiente” (tecnicamente in **spazio logaritmico**, ma per questa presentazione pensate pure ad un algoritmo polinomiale) tale che

$$\forall x (x \in A \leftrightarrow f(x) \in B)$$

E' evidente che se $B \in P$ e $A \leq B$ allora $A \in P$.
Similmente, se $B \in NP$ e $A \leq B$ allora $A \in NP$.

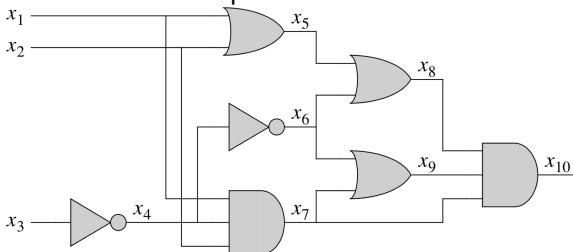
Un problema/linguaggio $L \in P$ è P completo se ogni linguaggio in P si può ridurre a lui. Un problema/linguaggio $L \in NP$ è NP completo se ogni linguaggio in NP si può ridurre a lui.

ARRIVANO COOK E LEVIN

CIRCUIT VALUE

Input: Un circuito logico con porte and, or, e not con una sola uscita e i valori fissati per gli ingressi. (e.g. $x_1 = 0, x_2 = 1, x_3 = 0$)

Problema: Stabilire se l'output è true.

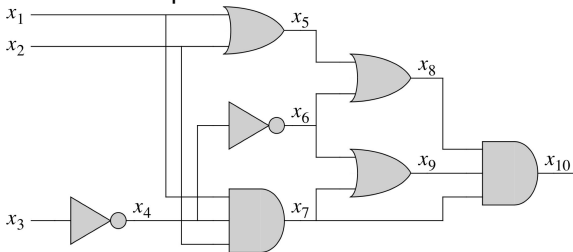


ARRIVANO COOK E LEVIN

CIRCUIT SAT

Input: Un circuito logico con porte and, or, e not con una sola uscita e n porte di ingresso.

Problema: Stabilire se esiste un assegnamento per le porte di ingresso che rende l'output true.



ARRIVANO COOK E LEVIN

THEOREM (1971)

CIRCUIT VALUE è P-completo. SAT è NP-completo.

- 1 Se un problema L sta in P esiste una macchina di Turing M che opera in tempo n^k per qualche k che lo decide.
- 2 Consideriamo la sua computazione come una matrice $n^k \times n^k$
- 3 La cella i della MdT al tempo $t + 1$ dipende solo dalle celle $i - 1, i, i + 1$ al tempo t e dalla posizione e stato della testina:
implementiamo quella matrice con un circuito logico
- 4 Otteniamo una istanza di Circuit Value
- 5 Se L stava in NP , parto da una MdT non deterministica e aggiungo un input esterno ad ogni livello. Ottengo una istanza di (CIRCUIT) SAT

P VERSUS NP

- E' evidente che $P \subseteq NP$.
- Ma sarà $P = NP$ o $P \subset NP$?
- Il problema è stabilire quale delle due è vera. Grazie a Cook-Levin:
- Per mostrare che $P = NP$ sarebbe sufficiente scrivere un algoritmo polinomiale che risolva SAT.
- Per mostrare che $P \subset NP$ ($P \neq NP$) bisognerebbe dimostrare che un tale algoritmo non esiste.
- E' un problema importante?

P VERSUS NP

- E' evidente che $P \subseteq NP$.
- Ma sarà $P = NP$ o $P \subset NP$?
- Il problema è stabilire quale delle due è vera. Grazie a Cook-Levin:
- Per mostrare che $P = NP$ sarebbe sufficiente scrivere un algoritmo polinomiale che risolva SAT.
- Per mostrare che $P \subset NP$ ($P \neq NP$) bisognerebbe dimostrare che un tale algoritmo non esiste.
- E' un problema importante?

I PROBLEMI DEL MILLENNIO

[HTTP://WWW.CLAYMATH.ORG/MILLENNIUM-PROBLEMS](http://www.claymath.org/millennium-problems)

- 1 Yang-Mills and Mass Gap
- 2 Riemann Hypothesis
- 3 **P vs NP Problem.** If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question.

If you prove that $P = NP$ you are able to solve mechanically all the others [Aaronson 2017]

- 4 Navier-Stokes Equation
- 5 Hodge Conjecture
- 6 Poincaré Conjecture (risolta da Grigoriy Perelman nel 2003)
- 7 Birch and Swinnerton-Dyer Conjecture

ON *P versus NP*

- Abbiamo visto Gödel e Von Neumann
- *Now my general conjecture is as follows: for almost all sufficiently complex types of enciphering . . . the mean key computation length increases exponentially with the length of the key, or in other words, the information content of the key . . . The nature of this conjecture is such that I cannot prove it, even for a special type of ciphers. Nor do I expect it to be proven.* [John Nash, 1955 (Nobel Economia 1994)]
- *P versus NP* — a gift to mathematics from computer science [Steve Smale (Fields 1996, Wolf 2006)]
- *The P versus NP problem deals with the central mystery of computation. The story of the long assault on this problem is our Iliad and our Odyssey; it is the defining myth of our field.* [Eric Allender, 2009]

ON *P versus NP*

- Juris Hartmanis (TA 1993). Gödel, von Neumann and the $P \stackrel{?}{=} NP$ Problem (1989).
- Michael Sipser. The History and Status of the P versus NP Question (1992).
- Stephen Cook (TA 1982). The Importance of the P versus NP Question (2003).
- Avi Widgerson. P, NP and mathematics — a computational complexity perspective (2006).
- Eric Allender. A Status Report on the P versus NP Question (2009).
- Scott Aaronson. $P \stackrel{?}{=} NP$ (2017).

SAT SOLVING



Martin Davis (1928)

Anche se Gödel e Turing hanno dimostrato che è impossibile l'automazione completa della logica del primo ordine, è comunque sensato affrontare l'automazione della ricerca di dimostrazioni di lunghezza finita (la parte “obviously, easily” della lettera di Gödel)



Hillary Putnam (1926–2016)

DAVIS PUTNAM — JACM 7(3):201–215 (1960)

A Computing Procedure for Quantification Theory*

MARTIN DAVIS

Rensselaer Polytechnic Institute, Hartford Division, East Windsor Hill, Conn.

AND

HILARY PUTNAM

Princeton University, Princeton, New Jersey

6. An Example

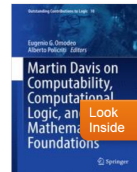
P. C. Gilmore¹⁸ tested his refutation-procedure on a number of formulas, including the following one:

$$(Ex)(Ey)(z)\{ (F(x, y) \rightarrow (F(y, z) \& F(z, z))) \& ((F(x, y) \& G(x, y)) \rightarrow (G(x, z) \& G(z, z))) \} \quad (1)$$

We have selected this example for purposes of comparison because (a) it is not so long as to make hand computation immediately impractical (e.g., it is already in prenex form, and the matrix can easily be put into conjunctive normal form); yet (b) Gilmore's procedure did *not* lead to a refutation although an IBM 704 was employed for 21 minutes.

Our procedure, on the other hand, *did* lead to a refutation in under a half-hour of *hand* computation!

¹⁸ Cf. PAUL C. GILMORE, A proof method for quantification theory. *IBM J. Research Dev.* 4 (1960), 28–35.



DAVIS PUTNAM — JACM 7(3):201–215 (1960)

A Computing Procedure for Quantification Theory*

MARTIN DAVIS

Rensselaer Polytechnic Institute, Hartford Division, East Windsor Hill, Conn.

AND

HILARY PUTNAM

Princeton University, Princeton, New Jersey

6. An Example

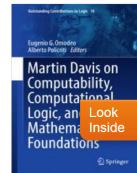
P. C. Gilmore¹⁸ tested his refutation-procedure on a number of formulas, including the following one:

$$(Ex)(Ey)(z)\{ (F(x, y) \rightarrow (F(y, z) \ \& \ F(z, z))) \ \& \ ((F(x, y) \ \& \ G(x, y)) \rightarrow (G(x, z) \ \& \ G(z, z))) \} \quad (1)$$

We have selected this example for purposes of comparison because (a) it is not so long as to make hand computation immediately impractical (e.g., it is already in prenex form, and the matrix can easily be put into conjunctive normal form); yet (b) Gilmore's procedure did *not* lead to a refutation although an IBM 704 was employed for 21 minutes.

Our procedure, on the other hand, *did* lead to a refutation in under a half-hour of hand computation!

¹⁸ Cf. PAUL C. GILMORE, A proof method for quantification theory. *IBM J. Research Dev.* 4 (1960), 28–35.



DAVIS PUTNAM — JACM 7(3):201–215 (1960)

Due ingredienti principali:

UNIT RULE Data una clausola

$$\ell_1 \vee \dots \vee \ell_{n-1} \vee \ell_n$$

tale che l'assegnamento finora calcolato per le variabili rende falsi $\ell_1, \dots, \ell_{n-1}$ allora ℓ_n va deterministicamente posto a vero.

RULE III (RESOLUTION) Riscrivi la formula

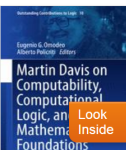
$$F = (\vec{a} \vee p) \wedge (\vec{b} \vee \neg p) \wedge R$$

con: $F' = (\vec{a} \vee \vec{b}) \wedge R$

RULE III* (SPLITTING) Choose a (free) variable x in F . Try $(x \wedge F)$. If it fails then try $(\neg x \wedge F)$
(Introduced by Loveland in [DPLL62])

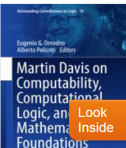
DPLL —PAGE 323 OF DAVIS' BOOK

The task of implementing the DP procedure was split. Logemann undertook the parsing of the CNF formula, entered in Polish notation, the formula preparation having been done by hand. Logemann also handled the structuring of the set of clauses, while Loveland took on the testing of the clause set for consistency. The program was written in SAP, the Symbolic Assembler Program, the assembly language for the IBM 704. After the first runs, which quickly saturated the 32,768 36-bit words of available storage, George Logemann suggested that Rule III be replaced with a splitting rule. He noted that it was easy to save the current environment on tape, and retrieve it on backtracking. As for the other systems with splitting rules, this led to depth-first search. Thus, with the new program, instead of clause addition there was clause removal. Interspersed with applications of Rules I and II, the program recursed on Rule III*, saving the environment on tape for backtracking. This solved the space issue, at least until the input clause set overwhelmed the memory. Now very important, but not emphasized at the time, was the easy definition of a satisfying assignment should there be one. It is unclear why “turning off” and then “turning on” various clauses and literals on backtracking was not pursued, ε is a slow operation. Using the appropriate list structures, top level that quickly deleted all occurrences of a specified literal and elimina



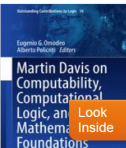
DPLL —PAGE 323 OF DAVIS' BOOK

The task of implementing the DP procedure was split. Logemann undertook the parsing of the CNF formula, entered in Polish notation, the formula preparation having been done by hand. Logemann also handled the structuring of the set of clauses, while Loveland took on the testing of the clause set for consistency. The program was written in SAP, the Symbolic Assembler Program, the assembly language for the IBM 704. After the first runs, which quickly saturated the 32,768 36-bit words of available storage, George Logemann suggested that Rule III be replaced with a splitting rule. He noted that it was easy to save the current environment on tape, and retrieve it on backtracking. As for the other systems with splitting rules, this led to depth-first search. Thus, with the new program, instead of clause addition there was clause removal. Interspersed with applications of Rules I and II, the program recursed on Rule III*, saving the environment on tape for backtracking. This solved the space issue, at least until the input clause set overwhelmed the memory. Now very important, but not emphasized at the time, was the easy definition of a satisfying assignment should there be one. It is unclear why “turning off” and then “turning on” various clauses and literals on backtracking was not pursued, as is a slow operation. Using the appropriate list structures, top level that quickly deleted all occurrences of a specified literal and elimina



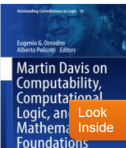
DPLL —PAGE 323 OF DAVIS' BOOK

The task of implementing the DP procedure was split. Logemann undertook the parsing of the CNF formula, entered in Polish notation, the formula preparation having been done by hand. Logemann also handled the structuring of the set of clauses, while Loveland took on the testing of the clause set for consistency. The program was written in SAP, the Symbolic Assembler Program, the assembly language for the IBM 704. After the first runs, which quickly saturated the 32,768 36-bit words of available storage, George Logemann suggested that Rule III be replaced with a splitting rule. He noted that it was easy to save the current environment on tape, and retrieve it on backtracking. As for the other systems with splitting rules, this led to depth-first search. Thus, with the new program, instead of clause addition there was clause removal. Interspersed with applications of Rules I and II, the program recursed on Rule III*, saving the environment on tape for backtracking. This solved the space issue, at least until the input clause set overwhelmed the memory. Now very important, but not emphasized at the time, was the easy definition of a satisfying assignment should there be one. It is unclear why “turning off” and then “turning on” various clauses and literals on backtracking was not pursued, as is a slow operation. Using the appropriate list structures, top level that quickly deleted all occurrences of a specified literal and elimina



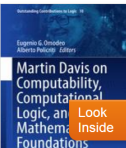
DPLL —PAGE 323 OF DAVIS' BOOK

The task of implementing the DP procedure was split. Logemann undertook the parsing of the CNF formula, entered in Polish notation, the formula preparation having been done by hand. Logemann also handled the structuring of the set of clauses, while Loveland took on the testing of the clause set for consistency. The program was written in SAP, the Symbolic Assembler Program, the assembly language for the IBM 704. After the first runs, which quickly saturated the 32,768 36-bit words of available storage, George Logemann suggested that Rule III be replaced with a splitting rule. He noted that it was easy to save the current environment on tape, and retrieve it on backtracking. As for the other systems with splitting rules, this led to depth-first search. Thus, with the new program, instead of clause addition there was clause removal. Interspersed with applications of Rules I and II, the program recursed on Rule III*, saving the environment on tape for backtracking. This solved the space issue, at least until the input clause set overwhelmed the memory. Now very important, but not emphasized at the time, was the easy definition of a satisfying assignment should there be one. It is unclear why “turning off” and then “turning on” various clauses and literals on backtracking was not pursued, as is a slow operation. Using the appropriate list structures, top level that quickly deleted all occurrences of a specified literal and elimina



DPLL —PAGE 323 OF DAVIS' BOOK

The task of implementing the DP procedure was split. Logemann undertook the parsing of the CNF formula, entered in Polish notation, the formula preparation having been done by hand. Logemann also handled the structuring of the set of clauses, while Loveland took on the testing of the clause set for consistency. The program was written in SAP, the Symbolic Assembler Program, the assembly language for the IBM 704. After the first runs, which quickly saturated the 32,768 36-bit words of available storage, George Logemann suggested that Rule III be replaced with a splitting rule. He noted that it was easy to save the current environment on tape, and retrieve it on backtracking. As for the other systems with splitting rules, this led to depth-first search. Thus, with the new program, instead of clause addition there was clause removal. Interspersed with applications of Rules I and II, the program recursed on Rule III*, saving the environment on tape for backtracking. This solved the space issue, at least until the input clause set overwhelmed the memory. Now very important, but not emphasized at the time, was the easy definition of a satisfying assignment should there be one. It is unclear why “turning off” and then “turning on” various clauses and literals on backtracking was not pursued, ϵ is a slow operation. Using the appropriate list structures, top level that quickly deleted all occurrences of a specified literal and elimina



SAT SOLVING

DPLL (1962)

Given Φ (CNF), establishing whether exists θ s.t. $\Phi\theta$ is true.

$DP(\Phi, \theta)$

$\theta' \leftarrow \text{unit_propagation}(\Phi, \theta)$

if (**satisfied** $(\Phi\theta')$) return θ'

else if (**failed** $(\Phi\theta')$) return **false**

else $X \leftarrow \text{select_variable}(\Phi, \theta')$

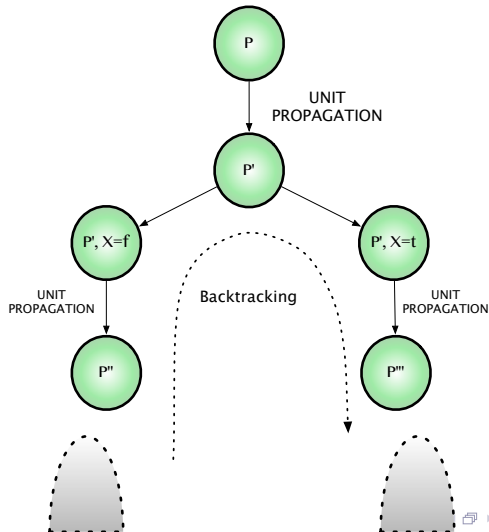
$\theta'' \leftarrow DP(\Phi, \theta'[X/\text{true}])$

if ($\theta'' \neq \text{false}$) return θ''

else return $DP(\Phi, \theta'[X/\text{false}])$

SAT SOLVING

DPLL (1962)



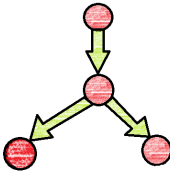
SEARCHING=PROPAGATION+ND ASSIGNMENT



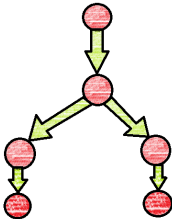
SEARCHING=PROPAGATION+ND ASSIGNMENT



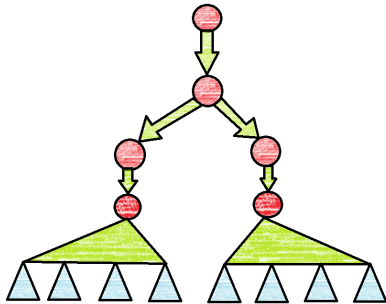
SEARCHING=PROPAGATION+ND ASSIGNMENT



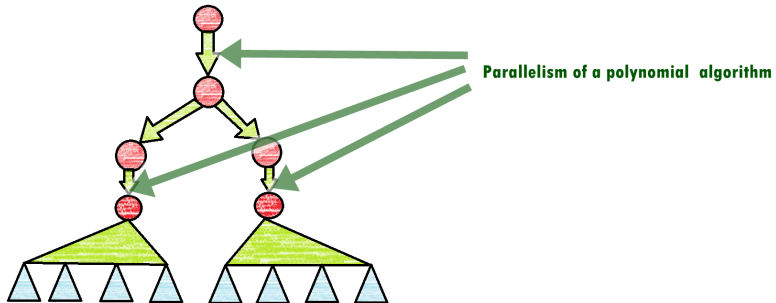
SEARCHING=PROPAGATION+ND ASSIGNMENT



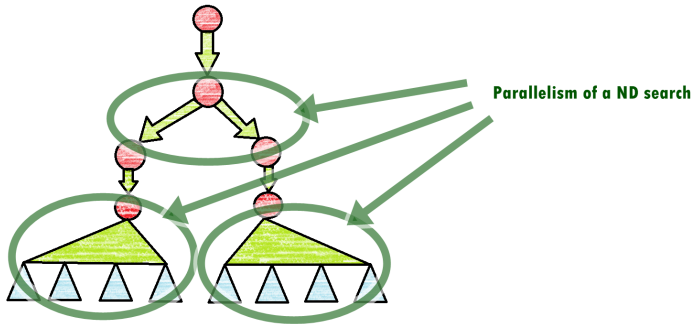
SEARCHING=PROPAGATION+ND ASSIGNMENT



SEARCHING=PROPAGATION+ND ASSIGNMENT



SEARCHING=PROPAGATION+ND ASSIGNMENT

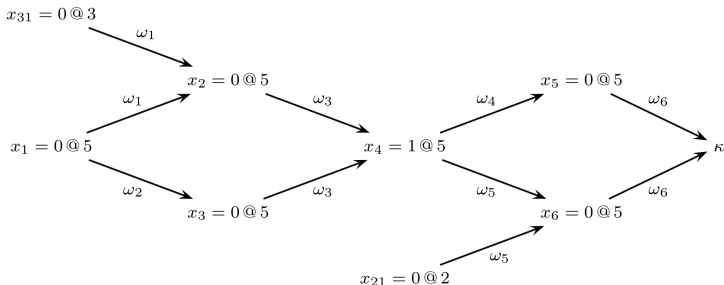


SAT COMPETITION

- Come abbiamo visto, SAT è il primo problema mostrato essere NP-completo
- Ogni problema in NP può essere ridotto a SAT
- Una soluzione efficiente per SAT sarebbe ereditata da tutti i problemi in NP
- Pertanto da DPLL in poi parte la progettazione dei cosiddetti **SAT SOLVERS**
- Un punto cruciale è la SAT COMPETITION
<http://www.satcompetition.org/> organizzata dal 2002.
- Un enorme speed-up avvenne con l'introduzione del **conflict-driven clause learning**.

CONFLICT DRIVEN CLAUSE LEARNING

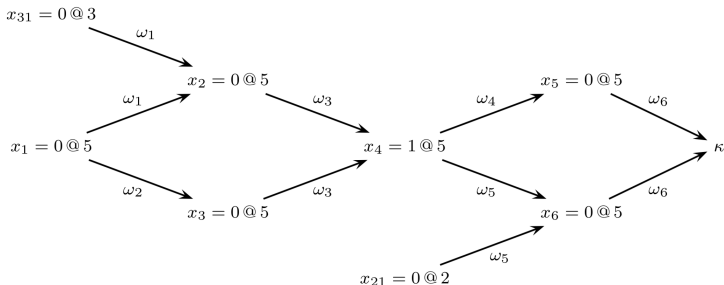
$$\overbrace{(x_1 \vee x_{31} \vee \neg x_2)}^{\omega_1} \wedge \overbrace{(x_1 \vee \neg x_3)}^{\omega_2} \wedge \overbrace{(x_2 \vee x_3 \vee x_4)}^{\omega_3} \wedge \overbrace{(\neg x_4 \vee \neg x_5)}^{\omega_4} \wedge \overbrace{(x_{21} \vee \neg x_4 \vee \neg x_6)}^{\omega_5} \wedge \overbrace{(x_5 \vee x_6)}^{\omega_6}$$



Possiamo *learn* $x_{31} \vee x_1 \vee x_{21}$. (e anche $\neg x_4 \vee x_{21}$ —diverse strategie vengono definite)
[Marques-Silva, Lynce, and Malik. Handbook of Satisfiability, Chapter 4, 2008]

CONFLICT DRIVEN CLAUSE LEARNING

$$\overbrace{(x_1 \vee x_{31} \vee \neg x_2)}^{\omega_1} \wedge \overbrace{(x_1 \vee \neg x_3)}^{\omega_2} \wedge \overbrace{(x_2 \vee x_3 \vee x_4)}^{\omega_3} \wedge \overbrace{(\neg x_4 \vee \neg x_5)}^{\omega_4} \wedge \overbrace{(x_{21} \vee \neg x_4 \vee \neg x_6)}^{\omega_5} \wedge \overbrace{(x_5 \vee x_6)}^{\omega_6}$$



Possiamo *learn* $x_{31} \vee x_1 \vee x_{21}$. (e anche $\neg x_4 \vee x_{21}$ —diverse strategie vengono definite)
[Marques-Silva, Lynce, and Malik. Handbook of Satisfiability, Chapter 4, 2008]

CONFLICT DRIVEN CLAUSE LEARNING

- Il grafo non viene scritto da qualche parte: viene scoperto applicando la *resolution rule* (quella di [DP60]):

$$(\alpha \vee X) \wedge (\beta \vee \neg X) \leftrightarrow (\alpha \vee \beta)$$

- Nell'esempio:

$$\begin{array}{c} \overbrace{(x_1 \vee x_{31} \vee \neg x_2)}^{\omega_1} \wedge \overbrace{(x_1 \vee \neg x_3)}^{\omega_2} \wedge \overbrace{(x_2 \vee x_3 \vee x_4)}^{\omega_3} \\ \overbrace{(\neg x_4 \vee \neg x_5)}^{\omega_4} \wedge \overbrace{(x_{21} \vee \neg x_4 \vee \neg x_6)}^{\omega_5} \wedge \overbrace{(x_5 \vee x_6)}^{\omega_6} \end{array}$$

$$X_6 : \omega_5 \wedge \omega_6 \leftrightarrow (X_{21} \vee \neg X_4 \vee X_5)$$

$$X_5 : (x_{21} \vee \neg x_4 \vee x_5) \wedge \omega_4 \leftrightarrow (x_{21} \vee \neg x_4) \text{ One could stop here or continue:}$$

$$X_4 : (x_{21} \vee \neg x_4) \wedge \omega_3 \leftrightarrow (X_{21} \vee X_2 \vee X_3)$$

$$X_3 : (X_{21} \vee X_2 \vee X_3) \wedge \omega_2 \leftrightarrow (X_{21} \vee X_1 \vee X_2)$$

$$X_2 : (X_{21} \vee X_1 \vee X_2) \wedge \omega_1 \leftrightarrow (X_{21} \vee x_{31} \vee x_1)$$

(now we have the decision variable of this level or variables of higher levels:
stop)

IL FORMATO DIMACS

- Per la SAT competition è stato fissato un formato standard.
- Le istanze sono in un file ASCII (con suffisso “.cnf”)
- Le variabili sono rappresentate da numeri interi (eccetto lo zero), con il $-$ si denota la negazione.
- I commenti iniziano con “c”, lo “0” termina le clausole
- C'è una linea iniziale iniziante con “p” che memorizza il numero di variabili e clausole.

$$(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_3)$$

```
c *****
c SAT Encoding of problem above
c *****
p cnf 3 2
1 -2 3 0
-1 -3 0
```


IDEA: RISOLVERE PROBLEMI NP USANDO SAT

- Devo affrontare un problema A che “mi sembra” NP
- Scrivo un programma nel mio linguaggio preferito che data una istanza del problema A la trasforma in una istanza di SAT “equivalente” (nel formato DIMACS)
- In pratica faccio una **riduzione** $A \leq SAT$
- Uso un SAT solver (dal sito della competizione ne posso scaricare diversi)
- Un grande classico è MiniSAT
- Se il problema da affrontare è NP-completo, non ci pentiremo!

ESEMPIO: SUDOKU

				1				
			2		3			
		4				5		
	6						7	
8				5				9
	1						3	
		5				4		
			7		1			
				9				

ESEMPIO: SUDOKU

- In ogni coppia (r, c) in $\{(1, 1), (1, 2), \dots, (9, 8), (9, 9)\}$ devo mettere uno ed un solo numero da 1 a 9.
- Introduco 9 variabili per ogni cella (r, c) : $X_1^{(r,c)}, \dots, X_9^{(r,c)}$
- Devo dire che almeno una di esse è vera:

$$X_1^{(r,c)} \vee X_2^{(r,c)} \vee X_3^{(r,c)} \vee X_4^{(r,c)} \vee X_5^{(r,c)} \vee X_6^{(r,c)} \vee X_7^{(r,c)} \vee X_8^{(r,c)} \vee X_9^{(r,c)}$$

- Devo dire che mai **due** di esse sono vere ovvero, per $i = 1, \dots, 8$ per $j = i + 1, \dots, 9$:

$$\neg X_i^{(r,c)} \vee \neg X_j^{(r,c)}$$

- Stiamo parlando di 9^3 variabili, dunque di uno spazio di ricerca di dimensione $2^{729} \approx 2.8 \cdot 10^{219}$

ESEMPIO: SUDOKU

- In ogni riga r per ogni valore k dall'1 al 9 c'è almeno una volta
- Devo dire che almeno una di esse è vera:

$$X_k^{(r,1)} \vee X_k^{(r,2)} \vee X_k^{(r,3)} \vee X_k^{(r,4)} \vee X_k^{(r,5)} \vee X_k^{(r,6)} \vee X_k^{(r,7)} \vee X_k^{(r,8)} \vee X_k^{(r,9)}$$

- Ma non due volte: Devo dire che mai **due** di esse sono vere
ovvero, per $i = 1, \dots, 8$ per $j = i + 1, \dots, 9$:

$$\neg X_k^{(r,i)} \vee \neg X_k^{(r,j)}$$

- Analogamente per le colonne (ve la lascio)

ESEMPIO: SUDOKU

0	0	0	1	1	1	2	2	2
0	0	0	1	1	1	2	2	2
0	0	0	1	1	1	2	2	2
3	3	3	4	4	4	5	5	5
3	3	3	4	4	4	5	5	5
3	3	3	4	4	4	5	5	5
6	6	6	7	7	7	8	8	8
6	6	6	7	7	7	8	8	8
6	6	6	7	7	7	8	8	8

Identifico i sottoquadrati

- Se le righe r vanno da 0 a 8 e le colonne c vanno da 0 a 8
- La cella (r, c) sta nel quadrato $3 * (r \text{ div } 3) + c \text{ div } 3$.
- Con questa idea si mettono i vincoli (analoghi ai precedenti) sulle variabili in ogni sottoquadrato.
Se r e c vanno da 1 a 9 (anziché da 0 a 8) basta mettere qualche "+1" e "-1" :)
- Quanto visto non dipende dalla specifica istanza!

ESEMPIO: SUDOKU

				1				
			2		3			
		4				5		
	6						7	
8				5				9
	1						3	
		5				4		
			7		1			
				9				

Devo esplicitare i fatti noti

- $X_1^{(1,5)}$
- $X_2^{(2,4)}$ e $X_3^{(2,6)}$
- ...
- $X_9^{(9,5)}$

ESEMPIO: SUDOKU

A questo punto dobbiamo solo trovare un mapping sensato tra

$$X_k^{(r,c)}$$

e un numero da 1 a 729 (per il DIMACS format)

Diamo un'occhiata a un codice in C e alla sua esecuzione.

Certo, tutto molto bello, ma scrivere/modificare una codifica in SAT può essere un incubo.

ESEMPIO: SUDOKU

A questo punto dobbiamo solo trovare un mapping sensato tra

$$X_k^{(r,c)}$$

e un numero da 1 a 729 (per il DIMACS format)

Diamo un'occhiata a un codice in C e alla sua esecuzione.

Certo, tutto molto bello, ma scrivere/modificare una codifica in SAT può essere un incubo.

ASP: UN LINGUAGGIO LOGICO PER NP

```

coord(1..9).
val(1..9).
% Assegno una coppia (X,Y) ad un sottoquadrato - e viceversa
square(I,X,Y) :- coord(X),coord(Y),coord(I),
    I == (X-1) / 3 + 3*((Y-1) / 3) + 1.
% Per ogni cella si assegna esattamente un valore
1 { x(X,Y,N) : val(N) } 1 :- coord(X), coord(Y).
% Ogni valore viene usato esattamente una volta in una colonna
1 { x(X,Y,N) : coord(X) } 1 :- coord(Y), val(N).
% Ogni valore viene usato esattamente una volta in una riga
1 { x(X,Y,N) : coord(Y) } 1 :- coord(X), val(N).
% Ogni valore viene usato esattamente una volta in un sottoquadrato
1 { x(X,Y,N) : square(I,X,Y) } 1 :- val(N), coord(I).
% istanza
x(1,5,1). x(2,4,2). x(2,6,3). ...

```

CONCLUSIONI

- Importanza teorica e pratica del problema SAT:
- Il problema P vs NP vi aspetta!
- Se avete un problema NP completo NON scrivete una soluzione ad-hoc ma trasformatelo in istanze di SAT (o ASP)
- I SAT solver (ma anche gli ASP solvers, i CP solvers, talvolta i tools della ricerca operativa basati sul semplice, talvolta i tools approssimati di ricerca locale) risolvono il vostro problema meglio di quello che potreste fare in diversi mesi.
- Tecniche di machine/deep learning vengono impiegate per la scelta della successiva variabile da testare e del valore (0 o 1) da provare per primo (metodi *portfolio*)
- Inoltre, se trovaste un algoritmo polinomiale per SAT (o se dimostraste che un tale algoritmo non può esistere ...)

QUESTIONS?

SAT A 2000M

