A l'intérieur d'un solveur SAT

François Schwarzentruber

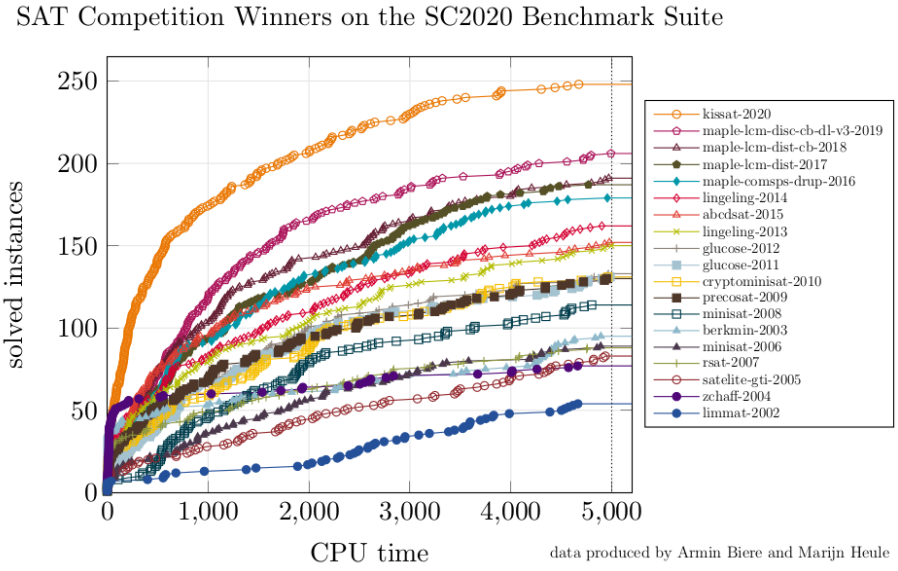
8 mai 2023

SAT
entrée : une formule φ en forme normale conjonctive
sortie : une valuation qui satisfait φ si φ est satisfiable ; unsat si φ est insatisfiable.

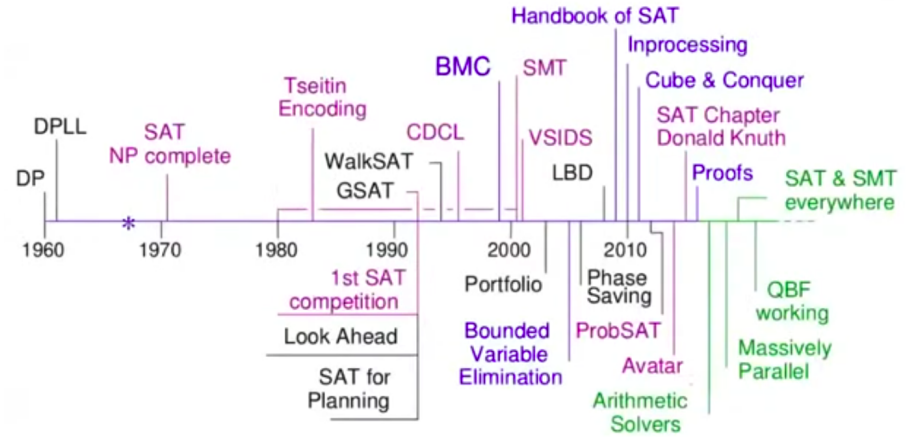
1 Applications

— bounded model checking
— planification automatique
— sous-routine pour des solveurs d'autres logiques (SMT, QBF, logique du premier ordre, ASP, etc.)
— résoudre des problèmes mathématiques (problème des triplets pythagoriciens booléens) [HKM16]

2 Performances des solveurs SAT



3 Historique



par Armin Biere

4 Notations

Soit AP un ensemble dénombrable de variables propositionnelles.

Définition 1 (valuation partielle) Une valuation partielle est une fonction partielle de AP dans $\{0,1\}$.

Remarque 2 Soit ℓ un littéral. $\neg\ell = \begin{cases} \neg p & \text{si } \ell = p \\ p & \text{si } \ell = \neg p \end{cases}$

Définition 3 Pour tout $i \in \{0,1\}$, on note $\begin{cases} \nu[p := i] & = \nu \cup \{(p,i)\} \\ \nu[\neg p := i] & = \nu \cup \{(p, 1-i)\}. \end{cases}$

Définition 4 Soit ℓ un littéral. On définit la notation $\nu \models \ell$ pour dire que $\begin{cases} (p,1) \in \nu & \text{si } \ell = p \\ (p,0) \in \nu & \text{si } \ell = \neg p \end{cases}$

Définition 5 (littéral assigné) ℓ est ν-assigné si $\nu \models \ell$ or $\nu \models \neg\ell$.

Définition 6 (condition de vérité d'une forme normale conjonctive)
- $\nu \models \bigwedge_{i\in\{1,\dots,n\}} \bigvee_{j\in\{1,\dots,k_i\}} \ell_{i,j}$ si pour tout $i \in \{1,\dots,n\}$, il existe $j \in \{1,\dots,k_i\}$ tel que $\nu \models \ell_{i,j}$.
- $\nu \models \neg \bigwedge_{i\in\{1,\dots,n\}} \bigvee_{j\in\{1,\dots,k_i\}} \ell_{i,j}$ si il existe $i \in \{1,\dots,n\}$ pour tout $j \in \{1,\dots,k_i\}$, on a $\nu \models \neg\ell_{i,j}$.

5 Backtracking

```
— entrée : une valuation partielle ν, une formule φ sous forme normale conjonctive
— sortie : true si ν peut être étendue en une valuation totale qui satisfait φ
fonction dpll(ν, φ)
    si ν ⊨ ¬φ alors renvoyer false
    si ν ⊨ φ alors renvoyer true
    sinon
        choisir une variable p non ν-assignée
        renvoyer dpll(ν[p := 0], φ) ou dpll(ν[p := 1], φ)
```

Exemple 7 $\overbrace{(p \vee q)}^{\alpha} \wedge \overbrace{(p \vee \neg q)}^{\beta} \wedge \overbrace{(\neg p \vee q)}^{\gamma} \wedge \overbrace{(\neg p \vee \neg q)}^{\delta} \wedge \overbrace{(r \vee s)}^{\epsilon}$.

6 Davis–Putnam–Logemann–Loveland (DPLL)

6.1 Clause unitaire

Définition 8 (clause unitaire) Une clause $\bigvee_j \ell_j$ est ν-unitaire s'il existe j_0 tel que pour tout $j \neq j_0$, $\nu \models \neg\ell_j$ et ℓ_{j_0} non ν-assigné.

$$\cancel{p} \vee \cancel{q} \vee \cancel{\neg s} \vee \ell.$$

6.2 Pseudo-code standard

```
fonction dpll(ν, φ)
    ν := propagationsUnitaires(ν, φ)
    si ν ⊨ ¬φ alors renvoyer false
    si ν ⊨ φ alors renvoyer true
    sinon
        choisir une variable p non ν-assignée
        renvoyer dpll(ν[p := 0], φ) ou dpll(ν[p := 1], φ)
```

```
— entrée : une valuation partielle ν, une CNF φ
— sortie : ν' qui étend ν telle que ν peut être étendue en une valuation satisfaisant φ
  ssi ν' peut être étendue en une valuation satisfaisant φ
fonction propagationsUnitaires(ν, φ)
    tant que il y a une clause ν-unitaire dans φ avec ℓ non ν-assigné
        ν := ν[ℓ := 1]
    renvoyer ν
```

6.3 Pseudo-code avec backtracking explicite

fonction $dpll(\nu, \varphi)$
  si $\nu = \bullet$ **alors renvoyer** false
  sinon
    $\nu :=$ propagationsUnitaires(ν, φ)
    si $\nu \models \neg\varphi$ **alors renvoyer** $dpll($ $backtrack(\nu, \varphi), \varphi$ $)$
    si $\nu \models \varphi$ **alors renvoyer** $true$
    sinon
      choisir une variable p non ν-assignée
      renvoyer $dpll(\nu[\ell := 1], \varphi)$

7 Conflict-driven clause learning (CDCL)

fonction cdcl(ν, φ)
  si $\nu = \bullet$ **alors renvoyer** false
  sinon
    $\nu :=$ propagationsUnitaires(ν, φ)
    si $\nu \models \neg\varphi$ **alors renvoyer** $dpll($ $backtrack - learning(\nu, \varphi)$ $)$
    si $\nu \models \varphi$ **alors renvoyer** $true$;
    sinon
      choisir une variable p non ν-assignée
      renvoyer $dpll(\nu[\ell := 1], \varphi)$

Exemple 9

$$\overbrace{(\neg x1 \vee x2)}^{\alpha} \wedge \overbrace{(\neg x1 \vee x5 \vee x3)}^{\beta} \wedge \overbrace{(x4 \vee \neg x2)}^{\gamma} \wedge \overbrace{(\neg x_3 \vee \neg x_4)}^{\delta} \wedge \overbrace{(x1 \vee x5 \vee \neg x2)}^{\epsilon} \wedge \overbrace{(x3 \vee x2)}^{\theta} \wedge \overbrace{(x2 \vee \neg x3)}^{\lambda} \wedge \overbrace{(\neg x5 \vee x6)}^{\mu} \wedge \overbrace{(x2)}^{\nu} .$$

Exemple 10

$$\overbrace{(x1 \vee x2)}^{\alpha} \wedge \overbrace{(\neg x2 \vee \neg x3)}^{\beta} \wedge \overbrace{(x3 \vee \neg x12)}^{\gamma} \wedge \overbrace{(\neg x2 \vee \neg x4 \vee \neg x5)}^{\delta} \wedge \overbrace{(x3 \vee x5 \vee x6 \vee x7)}^{\epsilon} \wedge \overbrace{(\neg x7 \vee x8 \vee \neg x9)}^{\theta} \wedge \overbrace{(x6 \vee \neg x7 \vee x9 \vee x10)}^{\lambda} \wedge \overbrace{(\neg x7 \vee \neg x10 \vee x8 \vee x11)}^{\mu} \wedge \overbrace{(x13 \vee \neg x14 < \vee \neg x15)}^{\nu} \wedge \overbrace{(x8 \vee \neg x7 \vee x11)}^{\rho} \wedge \overbrace{(\neg x11 \vee \neg x13)}^{\sigma} \wedge \overbrace{(\neg x11 \vee x14)}^{\tau} \wedge \overbrace{(x12 \vee x15)}^{\chi}$$

8 Améliorations

8.1 Redémarrage

— redémarrer après un certain nombre de backtracking, avec un seuil qui augmente (sinon on ne serait pas complet !)

8.2 Heuristiques

— Choix de la variable à décider, qui apparaît le plus fréquemment dans les clauses (Chaff)
— priorité aux variables qui apparaissent dans les clauses les plus récentes
— utilisation du machine learning

8.3 Suppression de clauses apprises

On peut apprendre un nombre exponentiel de clauses...
— On apprend que des clauses assez courtes (moins de n littéraux où n est fixé)
— On oublie une clause dès qu'au moins m littéraux sont non assignés

9 Recherche locale

fonction GSAT(φ)
  ν := valuation au hasard
  pour $i := 1$ à N **faire**
    si $\nu \models \varphi$ **alors renvoyer** ν
    v := variable parmi lesquelles, l'échange de la valeur $\nu(v)$ donne le plus de clauses satisfaites
    $\nu(v) := 1 - \nu(v)$
  renvoyer échec

Définition 11 Le break-count d'une variable est le nombre de clauses sont actuellement satisfaites mais qui deviennent insatisfaites si la variable est flipée.

fonction walkSAT(φ)
  ν := valuation au hasard
  pour $i := 1$ à N **faire**
    si $\nu \models \varphi$ **alors renvoyer** ν
    C := une clause non satisfaite choisie au hasard
    si il existe une variable x dans C avec break-count = 0 **alors** $v := x$
    sinon
      avec probabilité p : v := une variable de C au hasard
      avec probabilité $1 - p$: v := une variable de C avec break-count le plus petit
    $\nu(v) := 1 - \nu(v)$
  renvoyer échec

10 Notes bibliographiques

Les éléments techniques de ce cours viennent de [KS08] et [Har09]. La terminaison de l'algorithme CDCL est très bien expliquée dans la thèse [ZM03].

Références

[Har09] John Harrison. *Handbook of Practical Logic and Automated Reasoning.* Cambridge University Press, 2009.

[HKM16] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016.

[KS08] Daniel Kroening and Ofer Strichman. *Decision procedures*, volume 5. Springer, 2008.

[ZM03] Lintao Zhang and Sharad Malik. *Searching for truth : techniques for satisfiability of boolean formulas.* PhD thesis, Princeton University Princeton, 2003.