Cours 7: solveurs SAT

- Modélisation par SAT et solveurs SAT
- DPLL et propagation de contraintes
- Apprentissage de clauses
- Retour en arrière généralisé et structure de données
- Heuristiques: choix des choix, redémarrage aléatoire

Modélisation par SAT et solveurs SAT

SAT est le problème NP-complet de base.

Théorème de Cook. Tout problème NP se réduit polynomialement à SAT

Pourrait on utiliser en pratique un solveur "universel" basé sur SAT ?

1996: plannification de mouvements de robots efficace via SAT

Traduction en satisfiabilité de formules de logique propositionnelle !

1998: bounded model checking: au lieu de raisonner symboliquement sur la correction de propriété, tester l'existence d'exécutions incorrectes formées de $n = 1, 2, 3, \ldots$ étapes (chercher les bugs au lieu de prouver la correction)

Traduction en satisfiabilité de formules de logique propositionnelle !

à la même époque: gain de plusieurs ordres de grandeur sur la taille des formules SAT "traitables": de qq centaines de variables à plusieurs millions

Modélisation par SAT et solveurs SAT

SAT est le problème NP-complet de base.

Théorème de Cook. Tout problème NP se réduit polynomialement à SAT

Pourrait on utiliser en pratique un solveur "universel" basé sur SAT ?

Aujourd'hui l'utilisation de solveurs SAT génériques concurrence les algo ad-hoc pour certains problèmes d'optimisation combinatoire...

Différentes extensions: MaxSAT, QSAT, SAT modulo théory,...

équilibre à trouver entre pouvoir d'expression et efficacité des solveurs !

On se concentre aujourd'hui sur l'algorithmique de SAT.

Modélisation par SAT et solveurs SAT

les compétitions SAT: `www.satcompetition.org`

organisées régulièrement, plusieurs dizaines de candidats
catégories: industrial, handmade, random; SAT+UNSAT, SAT, UNSAT

Pas de solveur parfait, les problèmes sont de "natures" différentes:
$\Rightarrow$ être le meilleur pour différents type de problème dans une catégorie

Les compétitions permettent de vérifier rapidement les solveurs récents

Cours 7: solveurs SAT

- Modélisation par SAT et solveurs SAT
- DPLL et propagation de contraintes
- Apprentissage de clauses
- Retour en arrière généralisé et structure de données
- Heuristiques: choix des choix, redémarrage aléatoire

Rappels: CNF-SAT $=$ existe-t-il A tq $\phi[A] = \emptyset$?

Formule CNF:
(formule de logique propositionnelle en forme normale conjonctive)

n variables booléennes: $x_1, x_2, \ldots, x_n$

littéral $=$ variable ($\ell = x_i$) ou négation de variable ($\ell = \bar{x}_i$)

clause $=$ disjonction de littéraux: $\ell_1 \vee \ldots \vee \ell_k$

formule $=$ conjonction (ou ensemble) de clauses: $C_1 \wedge \ldots \wedge C_k$

assignation partielle $A =$ assignation de valeur $\textsc{vrai}$ ou $\textsc{faux}$ à un sous-ensemble des variables: exemple $\{x_1 = \textsc{vrai}, x_5 = \textsc{faux}\}$

l'évaluation $\phi[A]$: simplifier la formule ϕ en fonction de A

$\longrightarrow$ éliminer les littéraux $\textsc{faux}$, détecter les contradictions (=clause vide)

$\longrightarrow$ éliminer les clauses contenant un littéral assigné à $\textsc{vrai}$

$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$ devient $x_1 \wedge (\neg x_1 \vee x_4)$
pour $A = \{x_2 = \textsc{false}, x_3 = \textsc{true}\}$

Rappels: CNF-SAT $=$ existe-t-il A tq $\phi[A] = \emptyset$?

Clause unitaire = clause dont tous les littéraux sont assignés FAUX sauf un

Règle de propagation: il faut satisfaire les clauses unitaires

$$(x_1 \vee x_2 \vee \neg x_3) \text{ implique } x_1 = \text{VRAI}.$$

Propagation des contraintes unitaires: tant qu'il reste une clause unitaire, assigner le littéral correspondant et itérer

$\longrightarrow$ soit on arrive à une contradiction

$\longrightarrow$ soit on satisfait toutes les clauses

$\longrightarrow$ soit on arrive à une formule simplifiée, sans clause unitaire

Propagation est un algo polynomial, linéaire si on gère bien les clauses

DPLL: backtracking + unit propagation

Cf: cours INF431 sur l'exploration

DPLL = recherche arborescente + propagation de contraintes unitaires

Davis, Putnam, Logemann and Loveland (early 60's)

$DPLL(\phi, A) =$

– [PROPAGER]: propager les contraintes unitaires

– [DIAGNOSTIC]: Si formule vide renvoyer SAT

- si conflit backtrack
- ou si plus de backtrack possible, renvoyer UNSAT

– [CHOISIR]: choisir un litteral x

- explorer $DPLL(\phi[A = \text{VRAI}], A \cup \{x = \text{VRAI}\})$
- explorer $DPLL(\phi[A = \text{FAUX}], A \cup \{x = \text{FAUX}\})$

Version itérative avec pile explicite pour défaire les propagations unitaires.

Complétude de DPLL et résolution

DPLL est un algorithme complet:

– s'il existe une assignation valide il la trouve

– s'il répond UNSAT, l'arbre d'exploration constitue une "preuve" de la non-satisfiabilité de la formule.

Système de preuve: considérons la règle de résolution:

$$(x \vee \alpha) \wedge (\neg x \vee \beta) \quad \vdash \quad (\alpha \vee \beta)$$

Cette règle est complète et cohérente: en particulier

– si UNSAT, on peut dériver une clause vide (ou $y \wedge \neg y$) par résolutions

Arbre d'exploration de DPLL sur une formule UNSAT

$\Rightarrow$ preuve par résolution arborescente de même taille

La preuve par résolution fournit un certificat de non satisfiabilité.

Cours 7: solveurs SAT

- Modélisation par SAT et solveurs SAT
- DPLL et propagation de contraintes
- Apprentissage de clauses
- Retour en arrière généralisé et structure de données
- Heuristiques: choix des choix, redémarrage aléatoire

Apprentissage de clauses

Durant l'exploration lorsqu'un conflit est détecté, avant le backtrack, on apprend une clause qui explique le conflit et évite de le répéter.

$$\phi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

supposons les choix déjà faits: $c = \text{FAUX}$, $f = \text{FAUX}$

$$\phi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

on décide $a = \text{FAUX}$ et on propage:

$$\phi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

$$\phi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

$$\phi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

on arrive à un conflit: $(\neg d \vee \neg e \vee f)$ après les choix $c = f = a = \text{FAUX}$

on en déduit que la clause $(a \vee c \vee f)$ est satisfaite dans toute solution

Plus généralement la clause de décision exhaustive est obtenue comme disjonction de toutes les décisions prises avant d'aboutir à un conflit.

Apprentissage de clauses

Les clauses apprises sont ajoutées au fur et à mesure à la formule avant de continuer l'exploration: on espère qu'elles provoqueront des conflits anticipés et donc l'élimination de sous-arbres de l'arbre d'exploration.

Dans la pratique on étudie un graphe d'implication pour analyser chaque conflit et trouver des clauses *meilleures* que la clause de décision exhaustive formée en utilisant toutes les décisions prises jusqu'ici.

Clause de décision: n'inclure dans la clause que les variables associées aux décisions liées au conflit (qui sont dans des chemins d'implication vers lui)

Plus généralement, chaque coupure du graphe d'implication donne une clause qui peut être apprise. Les plus utiles avec la clause de décision sont:
– la clause de premier point d'implication unique (cf PC, exo 1),
– et la clause de plus proche coupure.

En pratique, le premier point d'implication unique fonctionne mieux.

Apprentissage de clauses et résolution

Les clauses apprises peuvent se déduire des autres par une séquence de résolutions élémentaires: on peut donc toujours déduire de l'échec de l'algorithme un certificat de non-satisfiabilité. Celui-ci s'obtient en temps linéaire en la taille de l'arbre de recherche et des clauses apprises.

On peut montrer que si on choisit bien les clauses qu'on apprend à chaque conflit (en apprenant notamment les clauses de plus proche coupure) l'algorithme est essentiellement aussi puissant que la résolution en général, alors que les résolutions arborescentes (et donc DPLL sans apprentissage) conduisent forcément sur certaines formules non satisfiables à des preuves de non satisfiabilité exponentiellement plus longues.

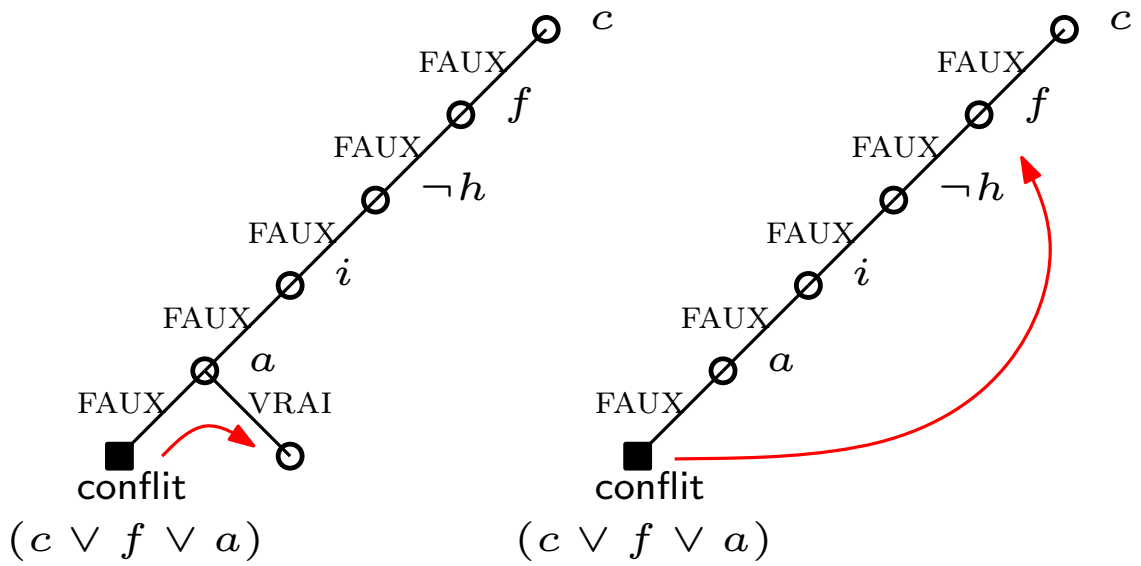
La taille des certificats (ou des preuves) est lié à la complexité dans le pire cas de la recherche d'une solution: en effet, imaginer une exploration qui part à la première décision dans une branche UNSAT.

Cours 7: solveurs SAT

- Modélisation par SAT et solveurs SAT
- DPLL et propagation de contraintes
- Apprentissage de clauses
- Retour en arrière généralisé et structure de données
- Heuristiques: choix de variables, redémarrage aléatoire

Retour en arrière généralisé

Supposons que la propagation après la décision $a = \text{FAUX}$ conduise à apprendre la clause $(c \vee f \vee a)$, sans que le conflit dépende de h et i



On peut revenir au choix de f directement !

En effet si on repart du choix $f = \text{FAUX}$, la clause apprise est unitaire et implique $a = \text{VRAI}$. (Pour que l'algorithme soit toujours complet il faut réexplorer la branche $f = \text{FAUX}$)

Cette approche nécessite un backtrack efficace (+ de retours en arrière).

Efficacité du backtrack: listes ou variables gardées

Par défaut on maintient:

– pour chaque clause: la liste des littéraux impliqués (par définition)

– pour chaque variable: liste des clauses la contenant (pour la propagation)

Remarques fondamentales: – l'algorithme reste correct et efficace même si on ne s'aperçoit pas tout de suite qu'une clause est satisfaite.

- par contre il faut trouver rapidement les clauses qui deviennent unitaires et les conflits

Lazy data structures: seules 2 variables par clause ont un pointeur de retour

– lorsqu'on assigne un littéral à FAUX, on cherche dans chaque clause pointée une variable remplaçante non assignée à FAUX: on détecte alors les nouvelles clauses unitaires et les conflits éventuels.

– lorsqu'on assigne un littéral à VRAI, on a rien à faire.

– lorsqu'on revient en arrière on a rien à faire !

– si toutes les variables sont assignées on sait que la formule est SAT

Cours 7: solveurs SAT

- Modélisation par SAT et solveurs SAT
- DPLL et propagation de contraintes
- Apprentissage de clauses
- Retour en arrière généralisé et structure de données
- Heuristiques: choix des choix, redémarrage aléatoire

Heuristiques: choix des choix, redémarrage aléatoire

Fail early! Instancier d'abord les variables qui apparaissent le plus souvent.

Avec les lazy data structures, on ne sait pas exactement combien il reste vraiment de clauses faisant intervenir une variable donnée.

$\Rightarrow$ règle très simple: un compteur associé à chaque variable, qui augmente chaque fois que cette variable apparaît dans une clause apprise;

instancier la variable ayant le compteur max.

Heavy tail distributions. Si on introduit une part de hasard dans le choix des choix, on constate une grande variabilité des temps d'exécution pour une même formule: la probabilité des exécutions catastrophiques ne décroît pas exponentiellement avec leur durée.

$\Rightarrow$ heuristique de redémarrage aléatoire: Fixer un temps limite T, et redémarrer le tout (éventuellement en conservant les clauses apprises) si le temps d'exploration dépasse T. Au bout de M redémarrages faire $T := 2T$ pour assurer la complétude.

Cours 7: solveurs SAT

- Modélisation par SAT et solveurs SAT
- DPLL et propagation de contraintes
- Apprentissage de clauses
- Retour en arrière généralisé et structure de données
- Heuristiques: choix des choix, redémarrage aléatoire

Retenir: les solveurs SAT sont surprenants d'efficacité

Modélisation plus ou moins facile, de gros succès (BMC,...)

Liens avec la théorie de la complexité des preuves

Des extensions MaxSAT, QSAT, Model counting...