Algorithmes de résolution

Une taxinomie des techniques de résolution

SAT Algorithmes

Complets

peut prouver l'insatisfiabilité

- Résolution
- Méthode des tableaux
- Procédure DPLL
- BDDs
- ...

Incomplets

Ne peut prouver l'insatisfiabilité

- Recherche locale
- Recuit simulé
- Recherche taboo
- Algorithmes génétiques
- ...

Algorithmes complets

- Méthodes syntaxiques : basé sur la résolution
- Méthodes sémantiques : énumératives (backtracking)

Méthodes basées sur la résolution

- ***Règle de résolution/consensus*** *:*

$$\frac{\omega_1 = (\neg a \vee \alpha),\ \omega_2 = (a \vee \beta)}{r = (\alpha \vee \beta)}$$

- *remarque : la résolvante* r est dite tautologique (vrai) si elle contient au moins deux littéraux opposés; sinon elle est dite fondamentale

- ***Règle de subsumption/sous-sommation*** *:*

si $\omega_1 \subseteq \omega_2$ alors $\omega_1 \models \omega_2$ (on dit que ω_1 subsume ω_2)

↳ soit Σ une formule CNF et $\omega_1, \omega_2 \in \Sigma$ tq. ω_1 subsume ω_2

alors Σ est satisfiable ssi $\Sigma \setminus \{\omega_2\}$ est satisfiable

- ***Règle de fusion*** *:*

$(a \vee \alpha) \models (a \vee \alpha \vee a)$ (cas particulier de la règle de subsumption)

Méthode de résolution

Fonction résolution (Σ)

tant que ($\bot \in S$) faire

choisir ω_1 , ω_2 $\in S$ contenant deux littéraux opposés

r = résolvante (ω_1, ω_2)

$S = S \cup \{r\}$

fin tant que

retourner faux

- La méthode résolution est complète pour la réfutation (*si la formule est insatisfiable alors la résolution produira la clause vide*)
- la résolution + la règle de subsumption (appliquée à chaque étape) est complète.

  soit Σ une formule satisfiable alors après un nombre fini d'étapes, toute nouvelle résolvante générée est subsumée par une clause de Σ

Procédure de Davis & Putnam [DP 60]

Procédure DP(Σ)

tant que ($\perp \notin \Sigma$ *et* $T \notin \Sigma$) *faire*

appliquer la règle de littéraux unitaire

appliquer la règle de littéraux pures

choisir une variable $p \in$ *Atomes*(Σ)

Soient $\Sigma_p = \{c \setminus \{p\}$, *tq.* $c \in \Sigma$, *et* $p \in c\}$,

$\Sigma_{\neg p} = \{c \setminus \{\neg p\}$, $c \in \Sigma$, *tq.* $\neg p \in c\}$ *et*

$\Sigma' = \{c \in \Sigma, p \notin c, \neg p \notin c\}$

$\Sigma = CNF(\Sigma_p \vee \Sigma_{\neg p}) \wedge \Sigma'$

fin tant que

si $\perp \in \Sigma$ *alors retourner faux*

sinon retourner vrai

Procédure de DP : un exemple

$\varphi = (a \vee c)\ (b \vee c)(d \vee c)(\neg a \vee \neg b \vee \neg c)$ Éliminer la variable *c*

$\varphi_1 = (a \vee \neg a \vee \neg b)(b \vee \neg a \vee \neg b\ )(d \vee \neg a \vee \neg b\ )$

$= (d \vee \neg a \vee \neg b\ )$ φ est satisfiable **!**

Méthode de résolution : sommaire

- exponentielle en temps et en espace
- preuve par réfutation ➔ rôle en logique du premier ordre
- adaptée à certaines classes de problèmes
- difficulté de mise en œuvre
- de nombreuses variantes (améliorations ont été proposées)
  - **SL-Résolution** [Kowalski-Kuehner:71]
  - Directional résolution [Dechter:00]
  - ...
- utilisation limitée en pratique
- des formes plus faibles sont souvent utilisées :
  - générer des résolvantes de taille réduite (< k)
    e.g. résolvantes binaires

Méthode des tableaux

- Recherche d 'un modèle de φ
  - appliquer récursivement les règles d'éliminations des connecteurs
  - si une branche contient A_i et $\neg A_i$ (ψ_i et $\neg\psi_i$) pour un certain i, la branche est dite fermée; sinon elle est dite ouverte
  - si aucune règle ne peut être appliquée à une branche ouverte, alors φ est satisfiable
  - si toute les branches sont fermées, alors φ est insatisfiable

Méthode des tableaux: règle d'élimination

$$\frac{\varphi_1 \wedge \varphi_2}{\begin{matrix}\varphi_1\\ \varphi_2\end{matrix}} \qquad \frac{\neg(\varphi_1 \vee \varphi_2)}{\begin{matrix}\neg\varphi_1\\ \neg\varphi_2\end{matrix}} \qquad \frac{\neg(\varphi_1 \rightarrow \varphi_2)}{\begin{matrix}\varphi_1\\ \neg\varphi_2\end{matrix}} \qquad \frac{(\varphi_1 \leftrightarrow \varphi_2)}{\begin{matrix}\varphi_1 \rightarrow \varphi_2\\ \varphi_2 \rightarrow \varphi_1\end{matrix}}$$

$\wedge$-élimination

$$\frac{\neg\neg\varphi}{\varphi}$$

$\neg$-élimination

$$\frac{\varphi_1 \vee \varphi_2}{\varphi_1 \quad \varphi_2} \qquad \frac{\neg(\varphi_1 \wedge \varphi_2)}{\neg\varphi_1 \quad \neg\varphi_2} \qquad \frac{\varphi_1 \rightarrow \varphi_2}{\neg\varphi_1 \quad \varphi_2}$$

$$\frac{\neg(\varphi_1 \leftrightarrow \varphi_2)}{\neg(\varphi_1 \rightarrow \varphi_2) \qquad \neg(\varphi_2 \rightarrow \varphi_1)}$$

$\vee$-élimination

Algorithme Tableau

Fonction Tableau (Γ) *\/*vrai si Γ est satisfiable; faux sinon*\/*

si $A_i \in \Gamma$ et $\neg A_i \in \Gamma$ *\/* branche fermée *\/*

alors retourner Faux

si $(\varphi_1 \wedge \varphi_2) \in \Gamma$ *\/* $\wedge$-élimination *\/*

alors retourner Tableau $(\Gamma \cup \{\varphi_1, \varphi_2\} \setminus \{ (\varphi_1 \wedge \varphi_2) \})$

si $(\neg\neg\, \varphi_1) \in \Gamma$ *\/* $\neg$ -élimination *\/*

alors retourner Tableau $(\Gamma \cup \{\varphi_1\} \setminus \{(\neg\neg\, \varphi_1)\})$

si $(\varphi_1 \vee \varphi_2) \in \Gamma$ *\/* $\vee$-élimination *\/*

alors retourner Tableau $(\Gamma \cup \{\varphi_1\} \setminus \{ (\varphi_1 \vee \varphi_2) \})$ ou

Tableau $(\Gamma \cup \{\varphi_2\} \setminus \{ (\varphi_1 \vee \varphi_2) \})$

retourner vrai

Méthode des tableaux : exemple

- $\varphi = \neg(\,(a \rightarrow b) \leftrightarrow (\neg a \vee b)\,)$

$\neg(\,(a \rightarrow b) \leftrightarrow (\neg a \vee b)\,)$

∨-élimination

$\neg(\,(a \rightarrow b) \rightarrow (\neg a \vee b)\,)$ $\qquad$ $\neg(\,(\neg a \vee b) \rightarrow (a \rightarrow b))$

∧-élimination $\qquad$ ∧-élimination

$(a \rightarrow b)\,,\ \neg(\neg a \vee b)$ $\qquad$ $(\neg a \vee b),\ \neg(a \rightarrow b)$

∨-élimination $\qquad$ ∨-élimination

$\neg a,\ \neg(\neg a \vee b)$ $\qquad$ $b,\ \neg(\neg a \vee b)$ $\qquad$ $\neg a,\ \neg(a \rightarrow b)$ $\qquad$ $b,\ \neg(a \rightarrow b)$

∧-élimination

$\neg a,\ \neg\neg a,\ \neg b$ $\qquad$ $b,\ \neg\neg a,\ \neg b$ $\qquad$ $\neg a,\ a,\ \neg b$ $\qquad$ $b,\ a,\ \neg b$

¬-élimination

$\neg a,\ a,\ \neg b$ $\qquad$ $b,\ a,\ \neg b$

Toute les branches sont fermées

↳ φ est insatisfiable

Méthode des tableaux : sommaire

- séparation sur les disjonctions (branchement)
- opère sur des formules quelconques
- intuitive, modulaire facile à étendre

  ↳ préférée par les logiciens

- inefficace ⇨pas très utilisée par les informaticiens
- nécessite un espace polynomiale

Algorithmes complets

- Méthodes syntaxiques : basé sur la résolution
- Méthodes sémantiques : énumératives (backtracking)

Procédure de DPLL [62]

- Procédure Davis-Putnam-Logeman-Loveland [DPLL] ou [DLL]

  construire récursivement un modèle de φ

  à chaque étape affecter une valeur à un atome

  effectuer les choix déterministes (obligatoires) en premier

- Règles de DPLL :

$$\frac{\varphi_1 \wedge (p) \wedge \varphi_2}{(\varphi_1 \wedge \varphi_2)[\,p|\top\,]} \quad \textbf{règle littéraux unitaire}$$

$$\frac{\varphi}{\varphi[\,p|\top\,]} \quad \textbf{p est pure}$$

$$\frac{\varphi}{\varphi[\,p|\top\,] \qquad \varphi[\,p|\bot\,]} \quad \textbf{règle séparation (split)}$$

Procédure de DPLL [62]

Fonction DPLL (φ, μ) **/*vrai si est satisfiable; faux sinon*/**

si φ = T **/* toute les clauses sont satisfaites */**
alors retourner vrai

si φ = $\perp$ **/* backtrak (retour arrière) */**
alors retourner faux

si (φ contient une clause unitaire (p)) **/* règle Unit */**

alors retourner DPLL ($\varphi[p|T]$, $\mu \wedge p$)

si (φ contient un littéral pure (p)) **/* règle pure */**
alors retourner DPLL ($\varphi[p|T]$, $\mu \wedge p$)

p := choix-littéral (φ) **/* Heuristique*/**

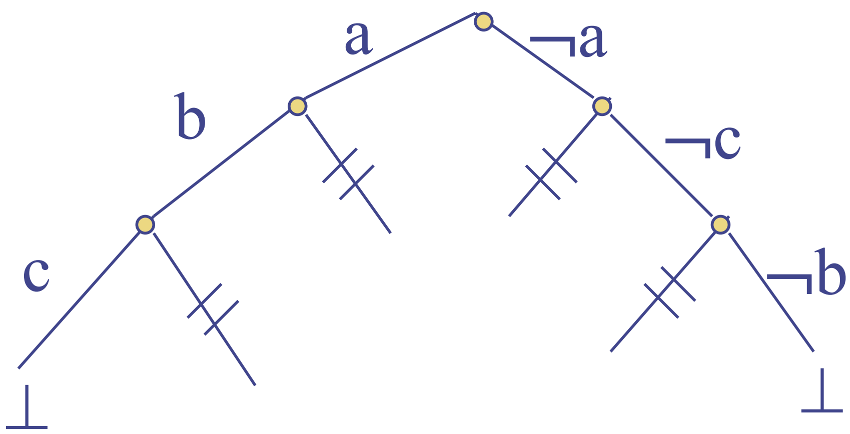
retourner DPLL ($\varphi[p|T]$, $\mu \wedge p$) $\vee$ DPLL ($\varphi[p|\perp]$, $\mu \wedge \neg p$) **/* split*/**

Procédure de DPLL : un exemple

- Soit **une formule CNF**

  $\varphi = (\neg a \vee \neg b \vee \neg c) \wedge (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a) \wedge (a \vee b \vee c)$

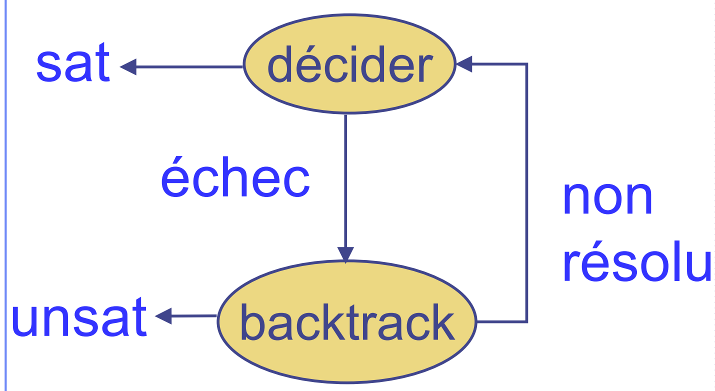
  - arbre binaire développé par DPLL
  - avec choix des atomes dans l 'ordre lexicographique

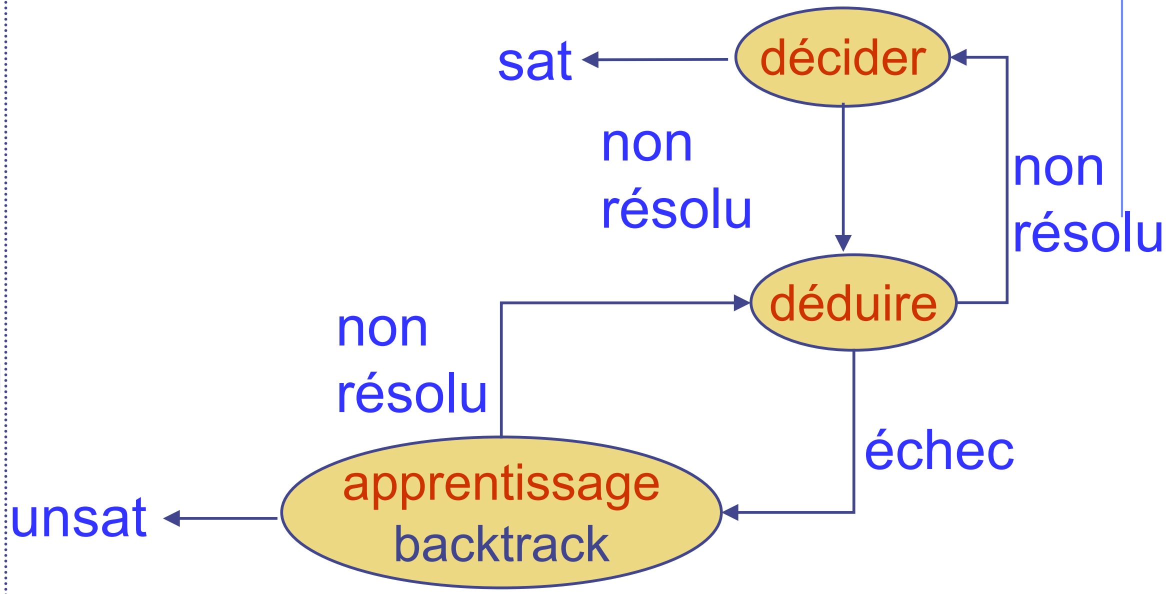


Procédure DPLL : sommaire

- Procède par énumération (affectation de valeurs aux atomes)
- retarde la séparation autant que possible
- nécessite une formule CNF en entrée (il existe des variantes pour des formules quelconques)
- ignorée par les logiciens
- la plus efficace des techniques de résolution de SAT
  - préférée par les informaticiens
- nécessite un espace polynomiale
- choix-littéral est critique pour l'efficacité !!
- de nombreuses variantes ont été proposées

DPLL : Heuristiques & optimisations





Recherche « naïve » → Recherche « intelligente »

Quine DPLL [Relsat , Grasp, Sato, Zchaff …]

[CSAT, Tableau, POSIT, Satz, ..]

Prospective (look-ahead) : détecter de futures situations d ’échecs

Rétrospective (look-back): apprendre des situations d ’échecs

DPLL : améliorations

- De nombreuses améliorations ont été proposées, elle concernent généralement un des points suivants :
  - **heuristiques de branchements** (choix de la prochaine variable à affecter)
    - heuristique de Jeroslow & wang
    - heuristique UP ...
  - **traitements des échecs « apprentissage »**
    - analyse de conflit
      - ajout de clauses «*clause recording*»
      - retour arrière non chronologique
        - «*conflict-directed backtraking* » (backjumping)
        - partition du modèle « *autarkness* » classique et généralisée ...
  - **simplifications** de la formule
    - effectuées sur la formule originale (pré-traitements)
      - résolution restreinte, 2-simplifications, suppression de clauses bloquées,...
    - effectuées en cours de résolution (traitement locaux)
      - traitement par propagation unitaire
      - exploitation des classes polynomiales, ...

Heuristiques de branchement

- La règle de séparation est source de non-déterminisme

  Soit φ une formule, et $p \in$ atomes(φ),

  $$\varphi \text{ est satisfiable ssi } \varphi[p|T] \vee \varphi[p|\perp] \text{ est satisfiable}$$

- Relation forte entre l'ordre d'affectation des variables et la taille de l 'arbre de recherche !

  ↳ De nombreuses heuristiques de choix de variables ont été proposées dans la littérature.

Heuristiques de branchement

- Une heuristique H est définie comme suit :

  $H : \text{Atomes}(\varphi) \rightarrow R$

  $x \rightarrow H(x) = f(\, w(x), w(\neg x)\,),$

  avec w(x) = poids associé à x, généralement calculé en utilisant des arguments syntaxique : longueur des clauses, nombre d'occurrences,...

  soit $\alpha = \max \{ H(x_i), x_i \in \text{atomes}(\varphi) \}$

  $\text{Choix}_H = \{x_k \in \text{Atomes}(\varphi), H(x_k) = \alpha\},$

  la variable de branchement x* est choisie dans Choix_H.

  ↳ - choix aléatoire,

  - utilisation d'une autre heuristique H' (*tie-breaker*)

Exemple d'heuristiques

- Utilisation du nombre d 'occurrences des littéraux :
  - w(x) = #occurrences du littéral x dans φ
  - w(x) = #occurrences du littéral x dans les clauses non-horn
  - w(x) = #occurrences du littéral x dans les clauses binaires,...
- Heuristique MOM : choix de la variable apparaissant le plus souvent dans les clauses les plus courts : « **M**aximum **O**ccurrences in clauses of **M**inimal length »
  - Jeroslow & Wang : choix d'un littéral avec un poids maximum

$$w(l) = \sum_{l \in c \,\&\, c \in \varphi} w(c)\,,\ avec\ w(c) = \frac{2^{n-|c|}}{\underbrace{2^{n}}} = 2^{-|c|}, n = \left|Atomes(\varphi)\right|$$

donne une estimation de la contribution de l à la satisfiabilité de φ

$$\frac{2^{n-|c|}}{2^{n}} \quad \frac{= |\ \{I,\ tq.\ I[c] = \perp\}\ |}{= \text{taille de l'espace de recherche}}$$

Exemple d'heuristiques

- Exemple de Hs :
  - $H1(x) = w(x) + w(\neg x)$ [Jeroslow &Wang]
  - $H2(x) = |w(x) - w(\neg x)|$
  - Introduction d 'un facteur d 'équilibre des arbres
    - $H3(x) = w(x) + w(\neg x) + \alpha \times \min(w(x), w(\neg x))$, $\alpha=1.5$ [Dubois-etal:93]
    - $H4(x) = w(x) + w(\neg x)+ \underbrace{\alpha \times w(x) \times w(\neg x),\ \alpha = 1024}$ [Freeman:96]

      - facteur d'équilibre
      - α constante empirique
  - Heuristique UP :[Freeman:96, Li:97]
    - $w(x) = |\varphi| - |\varphi'|$, avec φ' la formule obtenue à partir $\varphi[x|\perp]$ par propagation unitaire, $\Rightarrow$ maximise l'effet de la propagation unitaire,

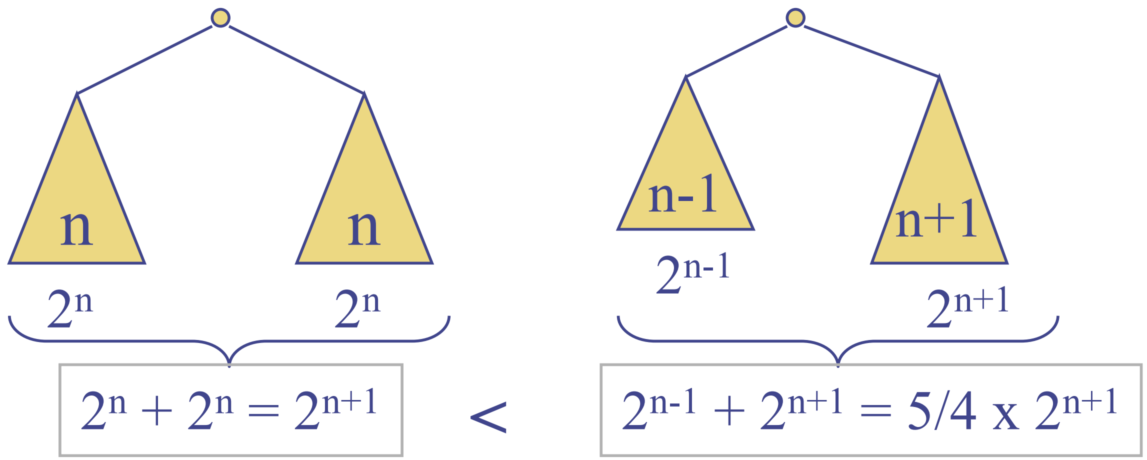
Exemple d'heuristiques

- soient $w(x)$ = #occurrences de x dans φ, x1, x2 $\in$ Atomes (φ) tq. $W(x1)= 6$, $w(\neg x1) = 6$ et $W(x2)= 8$, $w(\neg x2) =4$ alors
  - $H1(x1) = 6+6 = 12$, $H1(x2) = 12$

    ↳ x1 ou x2 ?
  - $H3(x1) = 6+6+\min(6,6) =18$, $H3(x2) = 8+4 + \min(8,4) = 16$

    ↳ *x1 est choisie*
  - $H4(x1) = 6+6+36 = 48$, $H4(x2) = 8+4+ 32 = 44$,

    ↳ x1 est choisie
- pourquoi équilibrer?



Traitement des échecs

- Idée : lorsqu'un conflit (contradiction) est détecté :

  1. déterminer un sous-ensemble de l'interprétation courante, responsable du conflit «*conflict-set* »
  2. Retour en arrière non-chronologique - saut au point responsable de l 'échec.

- l'ensemble conflit est construit à partir de la clause falsifiée en ne gardant que les littéraux affectés lors des points de choix.
- À chaque point de choix, un ensemble conflit est obtenu par résolution entre les ensembles conflits des deux branches.

↳Permet d'éviter de nombreuses explorations redondantes.

Traitement des échecs : ajout de clauses

- Au cours de la recherche, créer pour chaque conflit une clause permettant d'éviter l'apparition future du même conflit

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

Hypothèse (points de choix) $I[c] = \perp$ et $I[f] = \perp$ ($\neg$c et $\neg$f)

Affecter $\neg$a et effectuer la PU : b, d et e

Conflit : $(\neg d \vee \neg e \vee f)$ est falsifié

$\boxed{\neg a \wedge \neg c \wedge \neg f \Rightarrow \neg\varphi}$

$\boxed{\varphi \Rightarrow a \vee c \vee f}$

ajout d'une nouvelle clause: $(a \vee c \vee f)$

Traitement des échecs : ajout de clause

- Clause dérivée d'un conflit peut aussi être obtenue par application restreinte de la résolution

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

résolution

$(a \vee c \vee d)$

$(a \vee e)$

$(a \vee c \vee \neg e \vee f)$

$(a \vee c \vee f)$

La clause $(a \vee c \vee f)$ permet d'éviter le conflit !
Clause unitaire : a

Traitement des échecs : ajout de clauses

- Le nombre de clauses ajoutées peut être très grand !

  ↳ limiter le nombre de clauses ajoutées.

  - pour chaque conflit, une clause est ajoutée
  - garder uniquement les clauses de taille $\leq K$
  - les plus grande clauses active sont supprimées

  ↳ le nombre de clauses ajoutées est polynomial en K

- Relevance-based learning
  - supprimer les clauses actives avec $\geq M$ littéraux non affectés

Retour arrière non chronologique
« *conflict-directed backtraking* » *(backjumping)*

- Au cours de la recherche, et en présence de conflits retour arrière vers l'une des causes du conflit.

$$\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f)$$
$$(a \vee c \vee f)(\neg a \vee g)(\neg g \vee b)(\neg h \vee j)(\neg i \vee k)\ldots$$

Hypothèse (points de choix) $\neg c$, $\neg f$, $\neg h$ et $\neg i$

L'affectation de $\neg a$ crée un conflit $\Rightarrow$ clause $(a \vee c \vee f)$ ajouté
$(a \vee c \vee f)$ implique a

Un autre conflit est obtenu : $(\neg d \vee \neg e \vee f)$ est insatisfiable

$a \wedge \neg c \wedge \neg f \Rightarrow \neg\varphi$

$\varphi \Rightarrow \neg a \vee c \vee f$

$\therefore$ nouvelle clause ajoutée: $(\neg a \vee c \vee f)$

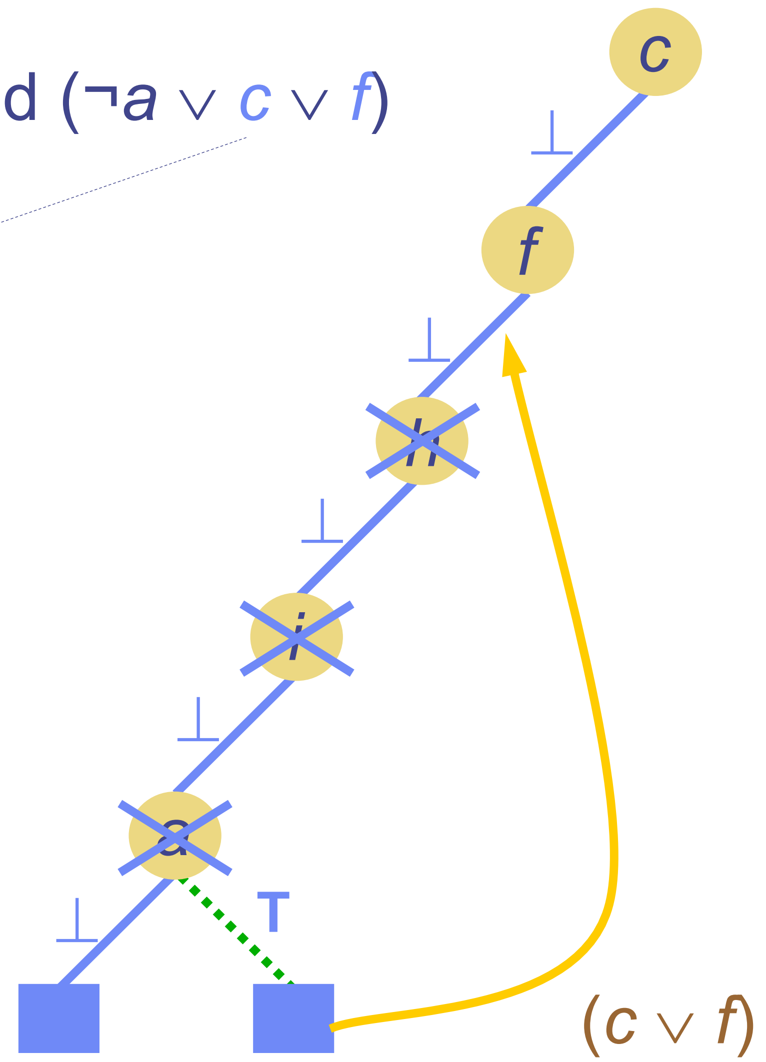
Retour arrière non chronologique
«*conflit-directed backtraking* » *(backjumping)*

Clauses ajoutées: $(a \vee c \vee f)$ and $(\neg a \vee c \vee f)$

Appliquer la résolution :
nouvelle clause falsifiée $(c \vee f)$

- retour au plus récent point de choix: $I[f] = \perp$

- clauses ajoutées:
  $(a \vee c \vee f)$,
  $(\neg a \vee c \vee f)$, et
  $(c \vee f)$



«*conflit-directed backtraking* » : Exemple

$$
\begin{aligned}
\varphi = \; & (\neg A_1 \vee A_2) \\
& (\neg A_1 \vee A_3 \vee A_9) \\
& (\neg A_2 \vee \neg A_3 \vee A_4) \\
& (\neg A_4 \vee A_5 \vee A_{10}) \\
& (\neg A_4 \vee A_6 \vee A_{11}) \\
& (\neg A_5 \vee \neg A_6) \\
& (A_1 \vee A_7 \vee \neg A_{12}) \\
& (A_1 \vee A_8) \\
& (\neg A_7 \vee \neg A_8 \vee \neg A_{13}) \\
& \ldots
\end{aligned}
$$

«*conflit-directed backtraking* » : Exemple (suite)

$\varphi = (\neg A_1 \vee A_2)$

$(\neg A_1 \vee A_3 \vee A_9)$

$(\neg A_2 \vee \neg A_3 \vee A_4)$

$(\neg A_4 \vee A_5 \vee A_{10})$

$(\neg A_4 \vee A_6 \vee A_{11})$

$(\neg A_5 \vee \neg A_6)$

$(A_1 \vee A_7 \vee \neg A_{12})$

$(A_1 \vee A_8)$

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$

…

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots\}$ **(Interprétation initiale)**

«*conflit-directed backtraking*» : Exemple (suite)

$\varphi = (\neg A_1 \vee A_2)$

$(\neg A_1 \vee A_3 \vee A_9)$

$(\neg A_2 \vee \neg A_3 \vee A_4)$

$(\neg A_4 \vee A_5 \vee A_{10})$

$(\neg A_4 \vee A_6 \vee A_{11})$

$(\neg A_5 \vee \neg A_6)$

$(A_1 \vee A_7 \vee \neg A_{12})$ ⟹ satisfaite

$(A_1 \vee A_8)$ ⟹ satisfaite

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$

…

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1\}$ (brancher sur A_1)

«*conflit-directed backtraking* » : Exemple (suite)

$\varphi = (\neg A_1 \vee A_2)$ ⇒ satisfaite

$(\neg A_1 \vee A_3 \vee A_9)$ ⇒ satisfaite

$(\neg A_2 \vee \neg A_3 \vee A_4)$

$(\neg A_4 \vee A_5 \vee A_{10})$

$(\neg A_4 \vee A_6 \vee A_{11})$

$(\neg A_5 \vee \neg A_6)$

$(A_1 \vee A_7 \vee \neg A_{12})$ ⇒ satisfaite

$(A_1 \vee A_8)$ ⇒ satisfaite

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$

…

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1, A_2, A_3\}$ (Unit A_2, A_3)

«*conflit-directed backtraking* » : Exemple (suite)

$\varphi =$ $(\neg A_1 \vee A_2)$ ⇒ satisfaite

$(\neg A_1 \vee A_3 \vee A_9)$ ⇒ satisfaite

$(\neg A_2 \vee \neg A_3 \vee A_4)$ ⇒ satisfaite

$(\neg A_4 \vee A_5 \vee A_{10})$

$(\neg A_4 \vee A_6 \vee A_{11})$

$(\neg A_5 \vee \neg A_6)$

$(A_1 \vee A_7 \vee \neg A_{12})$ ⇒ satisfaite

$(A_1 \vee A_8)$ ⇒ satisfaite

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$

…

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1, A_2, A_3, A_4\}$ (Unit A_4)

«*conflit-directed backtraking* » : Exemple (suite)

$\varphi = (\neg A_1 \vee A_2)$ ⟹ satisfaite

$(\neg A_1 \vee A_3 \vee A_9)$ ⟹ satisfaite

$(\neg A_2 \vee \neg A_3 \vee A_4)$ ⟹ satisfaite

$(\neg A_4 \vee A_5 \vee A_{10})$ ⟹ satisfaite

$(\neg A_4 \vee A_6 \vee A_{11})$ ⟹ satisfaite

$(\neg A_5 \vee \neg A_6)$ ⟹ conflit

$(A_1 \vee A_7 \vee \neg A_{12})$ ⟹ satisfaite

$(A_1 \vee A_8)$ ⟹ satisfaite

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$

…

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1, A_2, A_3, A_4, A_5, A_6, \}$ (Unit A_5, A_6)

« *conflit-directed backtraking* » : Exemple (suite)

$\varphi = (\neg A_1 \vee A_2)$ ⟹ satisfaite

$(\neg A_1 \vee A_3 \vee A_9)$ ⟹ satisfaite

$(\neg A_2 \vee \neg A_3 \vee A_4)$ ⟹ satisfaite

$(\neg A_4 \vee A_5 \vee A_{10})$ ⟹ satisfaite

$(\neg A_4 \vee A_6 \vee A_{11})$ ⟹ satisfaite

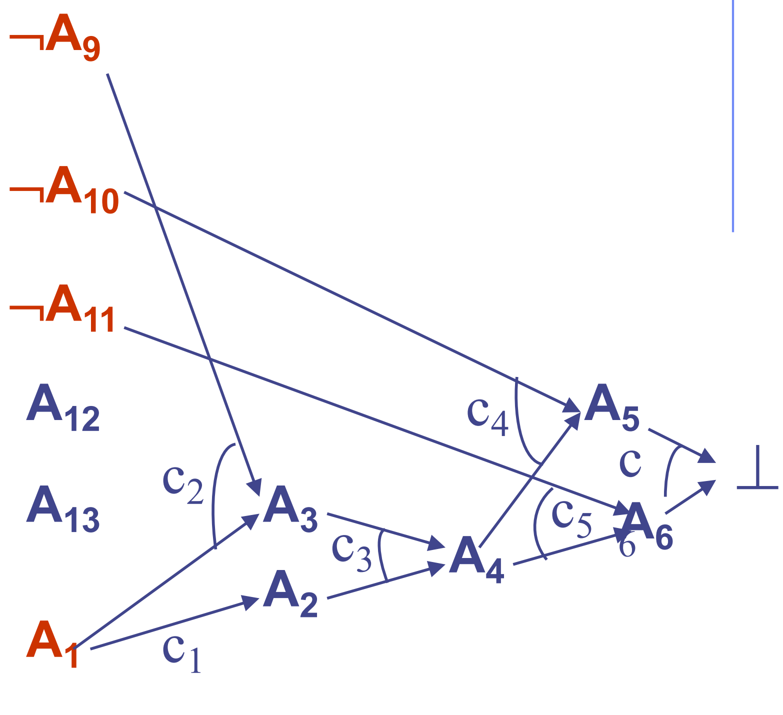
$(\neg A_5 \vee \neg A_6)$ ⟹ conflit

$(A_1 \vee A_7 \vee \neg A_{12})$ ⟹ satisfaite

$(A_1 \vee A_8)$ ⟹ satisfaite

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$ ⟹ satisfaite

...



Graphe d'implication

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1, A_2, A_3, A_4, A_5, A_6, \}$

ensemble conflit : $\{\neg A_9, \neg A_{10}, \neg A_{11}, A_1\}$

$(\neg A_9 \wedge \neg A_{10} \wedge \neg A_{11} \wedge A_1) \Rightarrow \neg\varphi$,

$\varphi \Rightarrow (A_9 \vee A_{10} \vee A_{11} \vee \neg A_1)$

∴ nouvelle clause ajoutée: $(A_9 \vee A_{10} \vee A_{11} \vee \neg A_1)$ **retour arrière sur A_1**

«*conflit-directed backtraking* » : Exemple (suite)

$\varphi =$

$(\neg A_1 \vee A_2)$ satisfaite

$(\neg A_1 \vee A_3 \vee A_9)$ satisfaite

$(\neg A_2 \vee \neg A_3 \vee A_4)$

$(\neg A_4 \vee A_5 \vee A_{10})$

$(\neg A_4 \vee A_6 \vee A_{11})$

$(\neg A_5 \vee \neg A_6)$

$(A_1 \vee A_7 \vee \neg A_{12})$

$(A_1 \vee A_8)$

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$

…

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, \neg A_1\}$ **(brancher sur $\neg A_1$)**

«*conflit-directed backtraking*» : Exemple (suite)

$\varphi = (\neg A_1 \vee A_2)$ ⟹ satisfaite

$(\neg A_1 \vee A_3 \vee A_9)$ ⟹ satisfaite

$(\neg A_2 \vee \neg A_3 \vee A_4)$

$(\neg A_4 \vee A_5 \vee A_{10})$

$(\neg A_4 \vee A_6 \vee A_{11})$

$(\neg A_5 \vee \neg A_6)$

$(A_1 \vee A_7 \vee \neg A_{12})$ ⟹ satisfaite

$(A_1 \vee A_8)$ ⟹ satisfaite

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$ ⟹ conflit

…

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, \neg A_1, A_7, A_8\}$(Unit A_7, A_8)

«*conflit-directed backtraking*» : Exemple (suite)

$\varphi = (\neg A_1 \vee A_2)$ ⟹ satisfaite

$(\neg A_1 \vee A_3 \vee A_9)$ ⟹ satisfaite

$(\neg A_2 \vee \neg A_3 \vee A_4)$

$(\neg A_4 \vee A_5 \vee A_{10})$

$(\neg A_4 \vee A_6 \vee A_{11})$

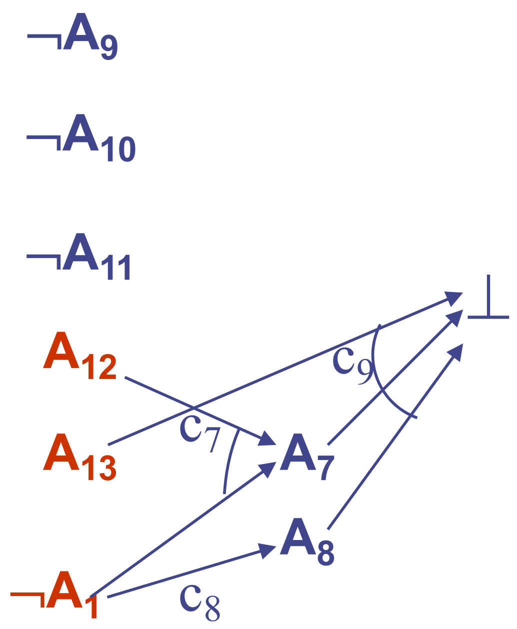
$(\neg A_5 \vee \neg A_6)$

$(A_1 \vee A_7 \vee \neg A_{12})$ ⟹ satisfaite

$(A_1 \vee A_8)$ ⟹ satisfaite

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$ ⟹ conflit

…



Graphe d'implication

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, \neg A_1, A_7, A_8\}$

➯ **ensemble conflit :** $\{A_{12}, A_{13}, \neg A_1\}$

$(A_{12} \wedge A_{13} \wedge \neg A_1) \Rightarrow \neg\varphi$,

$\varphi \Rightarrow (\neg A_{12} \vee \neg A_{13} \vee A_1)$ ∴ nouvelle clause ajoutée: $(\neg A_{12} \vee \neg A_{13} \vee A_1)$

« *conflit-directed backtraking* » : Exemple (suite)

$\varphi = (\neg A_1 \vee A_2)$ ⟹ satisfaite

$(\neg A_1 \vee A_3 \vee A_9)$ ⟹ satisfaite

$(\neg A_2 \vee \neg A_3 \vee A_4)$

$(\neg A_4 \vee A_5 \vee A_{10})$

$(\neg A_4 \vee A_6 \vee A_{11})$

$(\neg A_5 \vee \neg A_6)$

$(A_1 \vee A_7 \vee \neg A_{12})$ ⟹ satisfaite

$(A_1 \vee A_8)$ ⟹ satisfaite

$(\neg A_7 \vee \neg A_8 \vee \neg A_{13})$ ⟹ conflit

…

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, \neg A_1, A_7, A_8\}$

- clauses ajoutées:

$$\left.\begin{array}{l}(A_9 \vee A_{10} \vee A_{11} \vee \neg A_1) \\ (\neg A_{12} \vee \neg A_{13} \vee A_1)\end{array} \models (A_9 \vee A_{10} \vee A_{11} \vee \neg A_{12} \vee \neg A_{13})\right\}$$ Retour arrière sur A_{13}

Simplifications

- effectuées sur la formule originale (pré-traitements)
  - **Résolution restreinte** :

    - effectuer toutes les résolutions de longueur <=K (en général k <=2)

    Exemple : - soient $\omega_1 = (\neg a \vee b \vee c)$, $\omega_2 = (a \vee b)$

    ajouter $r = (b \vee c)$

    - soient $\omega_1 = (\neg a \vee b)$, $\omega_2 = (a \vee b)$

    ajouter $r = (b)$

    - effectuer des résolutions entre ω_1 et ω_2 si $|r| <= \max(|\omega_1|, |\omega_2|)$

    Exemple : soient $\omega_1 = (\neg a \vee b \vee c)$, $\omega_2 = (a \vee d)$

    ajouter $r = (b \vee c \vee d)$

Simplifications

- Eliminer des clauses et des variables
  - si $(x \vee \neg y)$, $(\neg x \vee y) \in \varphi$, alors x et y sont équivalent, $(x \leftrightarrow y)$
    - eliminer y, et en le remplaçant par x
    - supprimer les clauses satisfaites
  - utiliser la sous-formule 2-CNF pour identifier des littéraux équivalents, des littéraux impliqués,

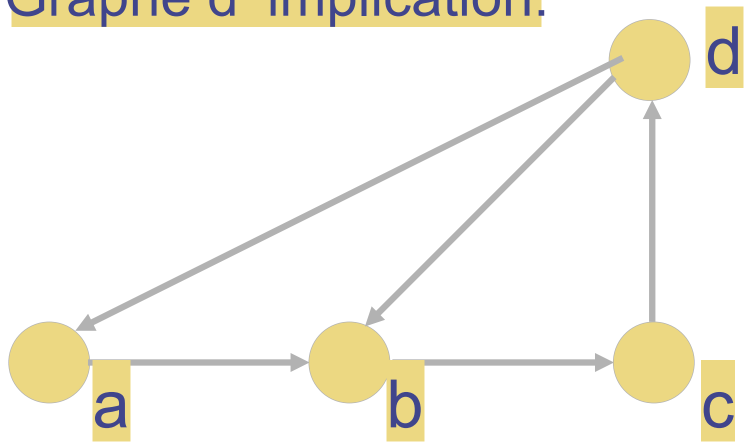
$(\neg a \vee b)(\neg b \vee c)(\neg c \vee d)(\neg d \vee b)(\neg d \vee a)$

$\equiv (a \rightarrow b)(b \rightarrow c)(c \rightarrow d)(d \rightarrow b)(d \rightarrow a)$

a, *b*, *c* et *d* sont équivalents deux à deux

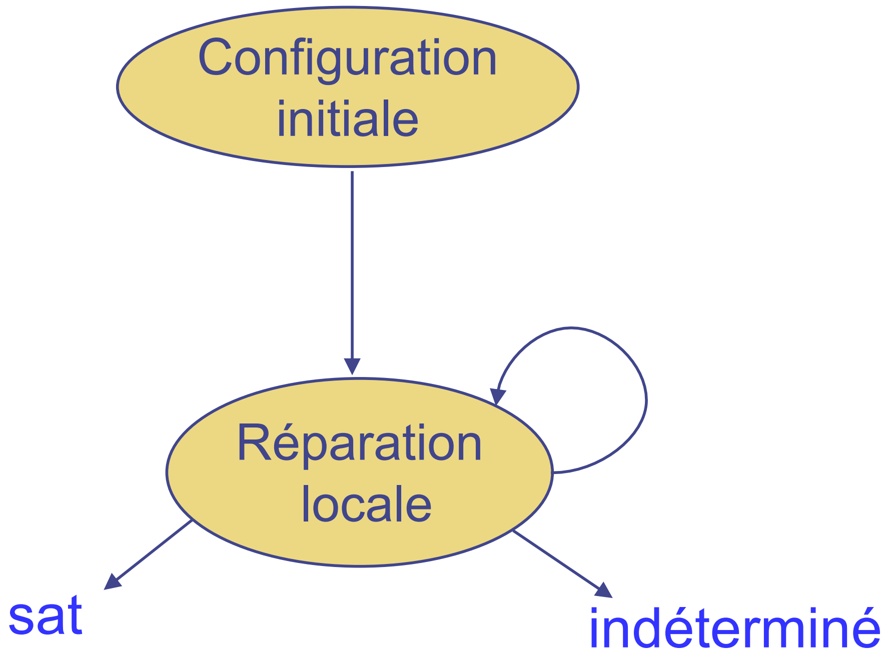
$\therefore$ remplacer toutes les variables par *a*

Graphe d'implication:



Algorithmes incomplets

Recherche locale (RL)



RO :

- Recuit simulé [Kirpatrick-et al.:83]
- Tabou [Glover:89]
- ...

SAT/ CSP :

- *Inversion method* [Dunham-Wang:76]
- SAT1 [Gu:87], QS [Minton:88]
- SAD [Hansen-Juamard:90] (Max-Sat)
- SCORE [Chabrier-et al.:91],
- GSAT [Selman-et al.:92] , ...

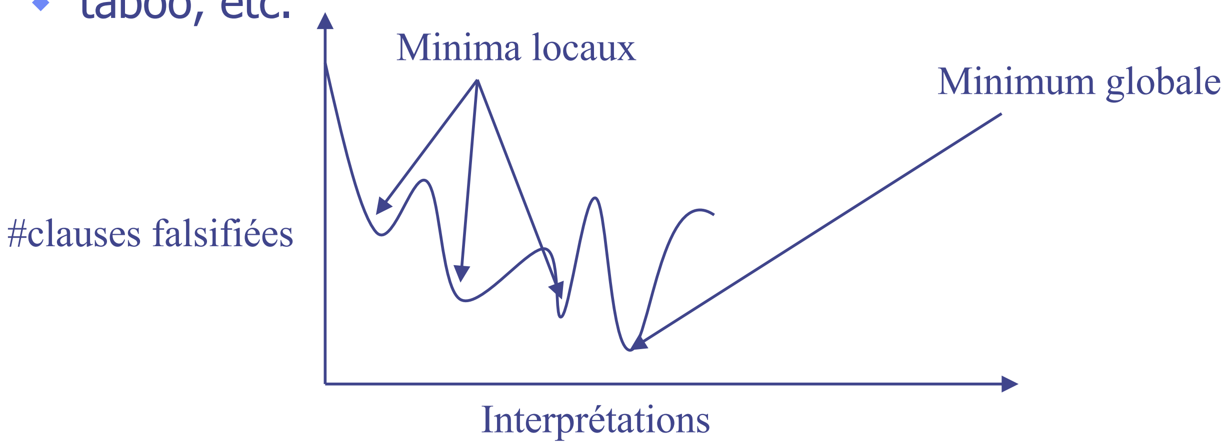
...

Recherche locale : définitions de base

- Espace de recherche : l'ensemble des interprétations (complète)
- Deux interprétations sont dites **voisines** si elle diffèrent sur la valeur d'une variable (distance de Hamming = 1).
- La **fonction d'évaluation d'une formule φ étant donnée une interprétation I** est définie par le nombre de clauses falsifiés par I, notée : (**score(φ, I)**)
- La **score d'une variable x étant donnée une interprétation I et une formule φ** (score(x, **φ,** I)) est défini par la différence entre le score de I et le score de I' (obtenue à partir de I en **inversant la valeur de vérité de x**)

Recherche locale : schéma de base

- Recherche non systématique d'une solution
  - génération aléatoire d'une interprétation (complète) initiale (1)
  - déplacement vers la meilleure interprétation « voisine »,
  - en essayant d 'éviter et/ou d 'échapper aux minima locaux :
    - recommencer en (1),
    - random walk,
    - taboo, etc.



L'algorithme GSAT

```
Fonction GSAT()
for i:= 1 to maxTries do
    μ:= randAssign(φ)
    for j:=1 to maxFlips do
        if (score(φ, μ) = 0)
        then return true;
        else bestFlips :=hillClimb(φ, μ);
            Ai := randPick(bestFlips);
            μ :=flip(Ai, μ);
    end
end
return « no satisfying assignement found »
```

GSAT : exemple

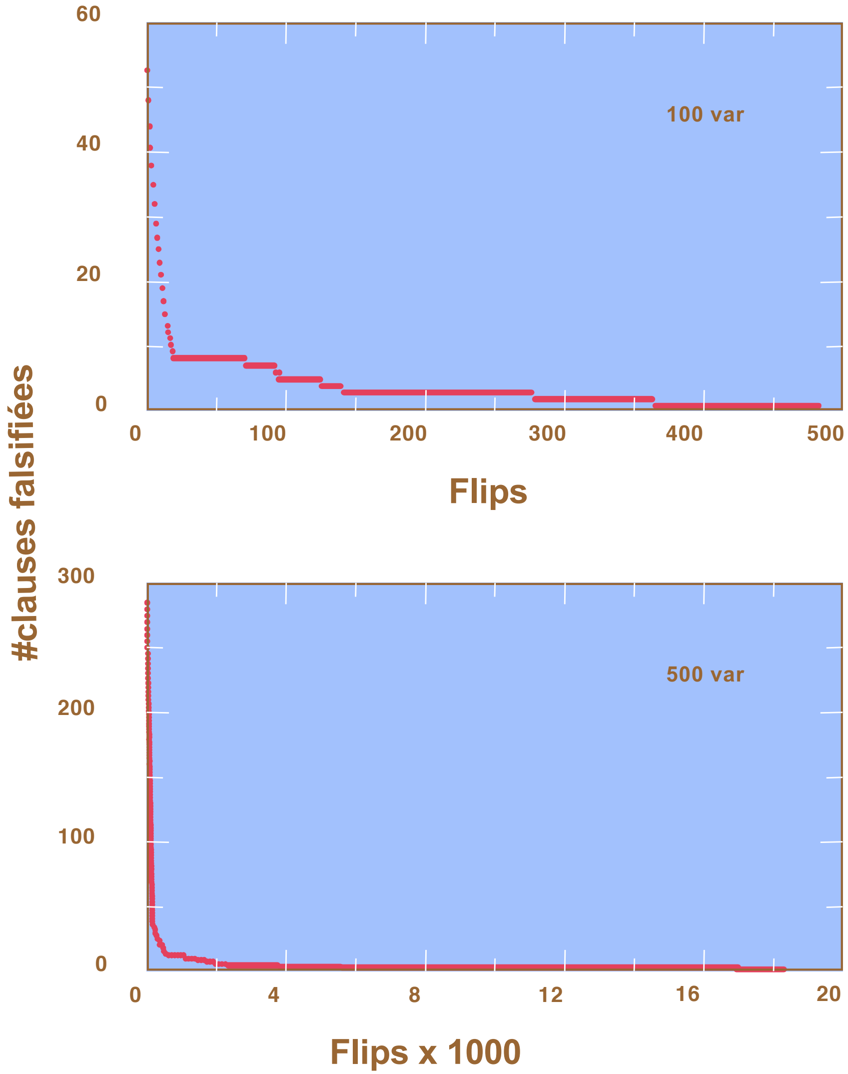
- Répéter maxTries fois:
  - générer aléatoirement une interprétation complète
  - répéter *maxFlips* fois (tant qu'il existe des clauses falsifiés):
    - « Flipper » une variable qui satisfait le maximum de clauses falsifiés (max>=0)

$\varphi = (a + b)(\neg a + c)\ (\neg b + d)(\neg c + d)$ Tirage aléatoire de I

$\varphi = (a + b)(\neg a + c)\ (\neg b + d)(\neg c + d)$ Flipper d dans I

$\varphi = (a + b)(\neg a + c)\ (\neg b + d)(\neg c + d)$ Instance est satisfiable !

GSAT : l 'espace de recherche



Remarque : pas de remontée

RL: Améliorations de l'algorithme de base

But: comment atteindre rapidement des plateaux (minima locaux) plus bas?

Stratégies d 'échappement des minima locaux :

a) Random Walk

(Selman, Kautz , and Cohen 1993)

b) Méthode tabou

(Fred Glover 1989)

Random Walk

- Random walk SAT algorithm:

  1) tirer aléatoirement une interprétation complète

  2) Répéter jusqu à satisfaire toute les clauses :

  Flipper aléatoirement une variable apparaissant dans une clauses falsifié

- Résout 2SAT en O(n) flips. (Papadimitriou 1992)
- Inefficace sur k-SAT (k >= 3).

Random Walk modifié (RWS)

1) Avec une probabilité, **"walk"** , i.e.,

flipper une variable apparaissant dans une clause falsifié.

2) Avce une probabilité 1-p, **"greedy move"** , i.e.,

flipper une variable satisfaisant le maximum de clauses

falsifiées.

Résultats : 3-SAT aléatoire au seuil

	GSAT base	GSAT RWS	Sim. Ann.
vars	temps(sc)	temps(sc)	temps(sc)
100	.4	.2	.6
200	22	4	21
400	122	7	75
600	1471	35	427
800	*	286	*
1000	*	1095	*
2000	*	3255	*

Random Walk > GSAT de base > DPLL

Autres variantes de GSAT

- HSAT : idem que GSAT de base : utilisation d 'une heuristique pour départager les variables ayant le score max :
  - donner moins de priorité à la variable la plus récemment « flippé »
- WSAT-G (p): tirer aléatoirement une clause falsifiée c :
  1. avec une probabilité p flipper aléatoirement une variable de c
  2. avec une probabilité 1-p, utiliser la stratégie de GSAT de base (choix de la variable satisfaisant le max de clauses)
- WSAT-B (p): idem que WSAT-G, sauf pour le cas 2 :
  - choix de la variable satisfaisant le maximum de nouvelles clauses

Autres variantes de GSAT

- Novelty (p) : tirer aléatoirement une clause c :
  1. flipper une variable xi de c ayant le meilleur score, sauf si xi est la variable la plus récemment flippé dans c.
  2. Dans ce dernier cas :
     - avec une probabilité p, flipper xi
     - avec une probabilité 1-p, flipper xk la deuxième variable de c ayant le meilleur score
- Novelty+ (p,q) : idem que Novelty , sauf le cas 1. à remplacer par :
  - avec une probabilité q flipper une variable xi de c (au hasard) ...
- ...

Tabou pour SAT

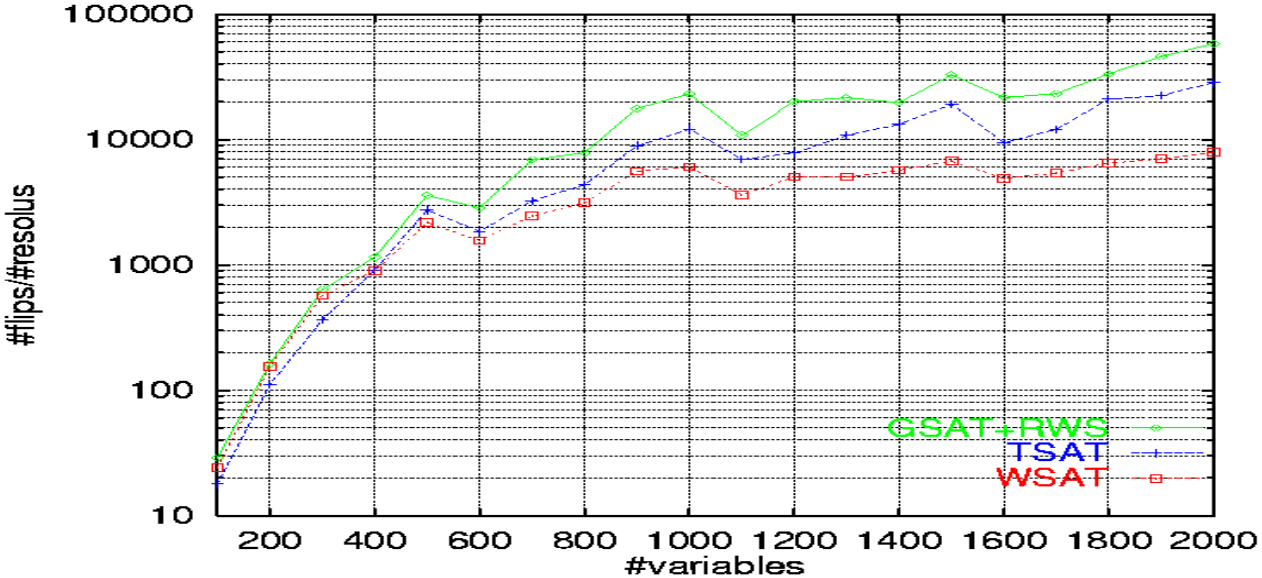
[Mazure-Saïs-Grégoire:97]

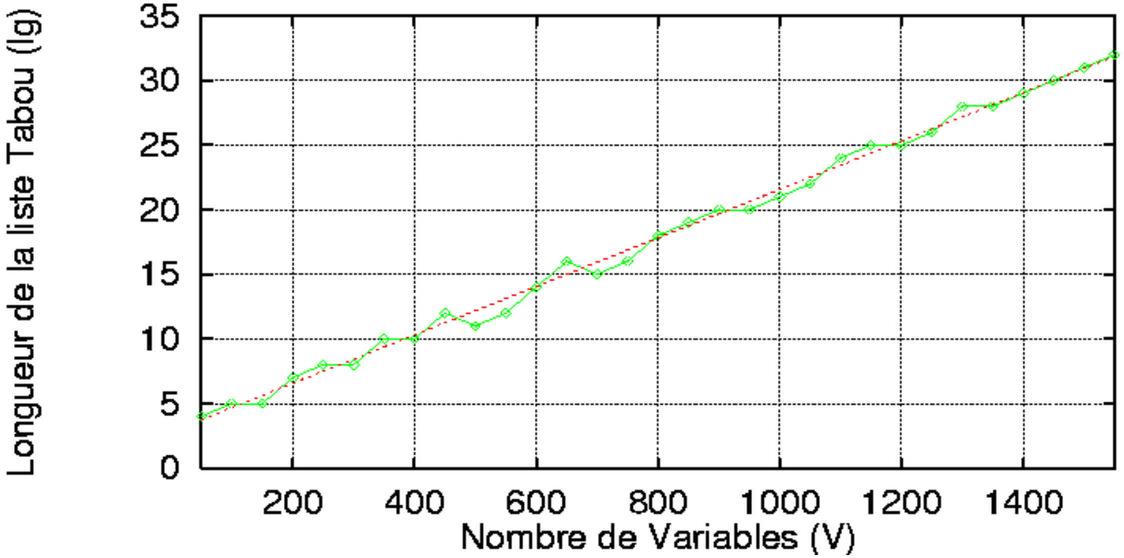
- Objectifs
  - Adapter efficacement des techniques de RL (RO) à la résolution de SAT
  - Supprimer le caractère aléatoire (prédominant) dans GSAT-RWS
- Quelle méthode? $\Rightarrow$ Tabou
  - TSAT = GSAT + liste d'interdits (tabou) de taille fixée

    la liste d 'interdits peut être vue comme une file d 'attente :
    - à chaque étape, il est interdit de choisir dans la liste Tabou la prochaine variable à flipper.
    - À chaque étape, une variable (flippée) entre dans la file (liste d'interdits) et une autre sort de la file.

  ↳ permet d'échapper des minima locaux, et d 'éviter des cycles,...
- Résultats
  - Meilleurs performances par rapport à GSAT-RWS
  - Taille optimale de la liste Tabou ?

3-SAT aléatoire (seuil)





GSAT+weight

- **Observation :**
  - pour certaines classes de problèmes, différentes exécutions de GSAT mènent vers les mêmes clauses falsifiées.
  - Ou encore vers une exploration répétée des mêmes parties de l'espace de recherche
  - ↳ mise en œuvre une stratégie pour éviter ce problème?
- **Réalisation :**
  - Pondération des clauses : associer un poids à chaque clause, initialement poids (c) =1 pour toute clause de c
  - Après chaque essai, incrémenter le poids des clauses falsifiées.
  - Modifier la fonction de calcul du score : le score d'une variable étant donnée une interprétation est pondéré par le poids des clauses

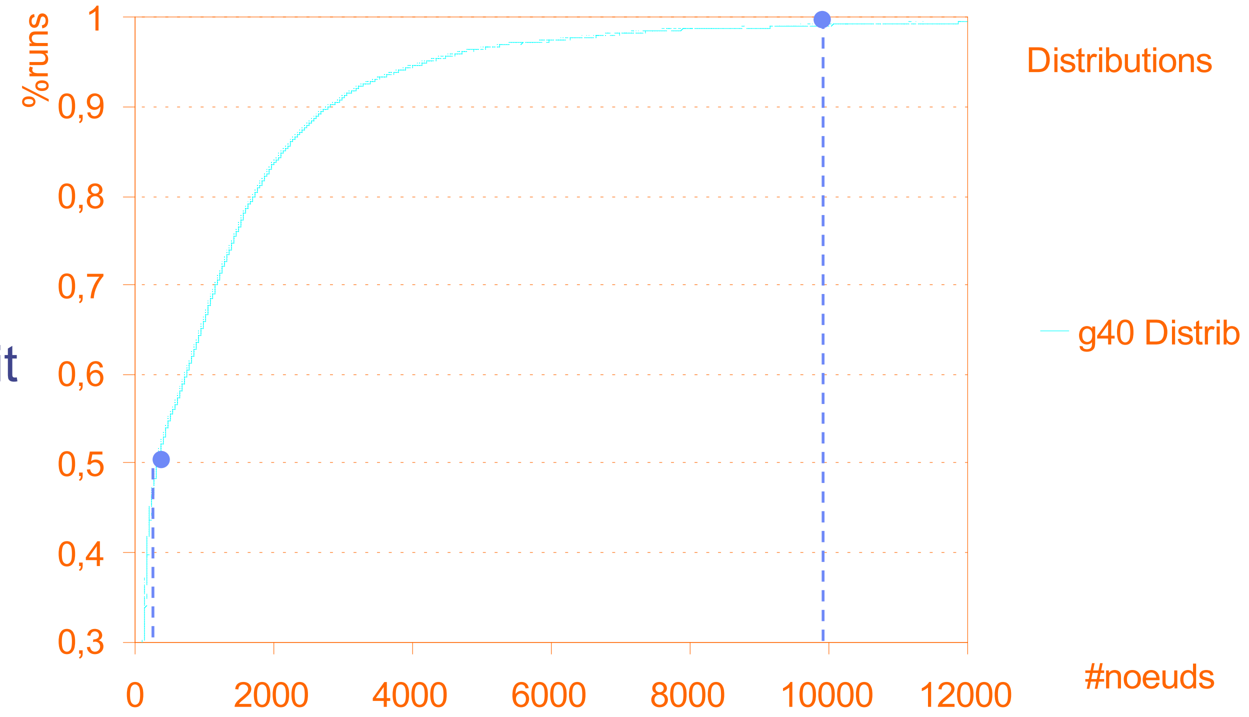
Recherche locale :sommaire

- Nécessite une formule CNF
- Incomplète : ne permet pas de prouver l 'insatisfiabilité
- très efficace sur certains problèmes (satisfiables)
- nécessite un espace polynomiale
- utilisé en intelligence artificielle (e.g. planification)
- de nombreuses variantes : GSAT-random walk, WSAT, ...
- variantes proposées pour les formules non-CNF : NC-GSAT, DAG-SAT

Autres méthodes : « Randomization & restart »

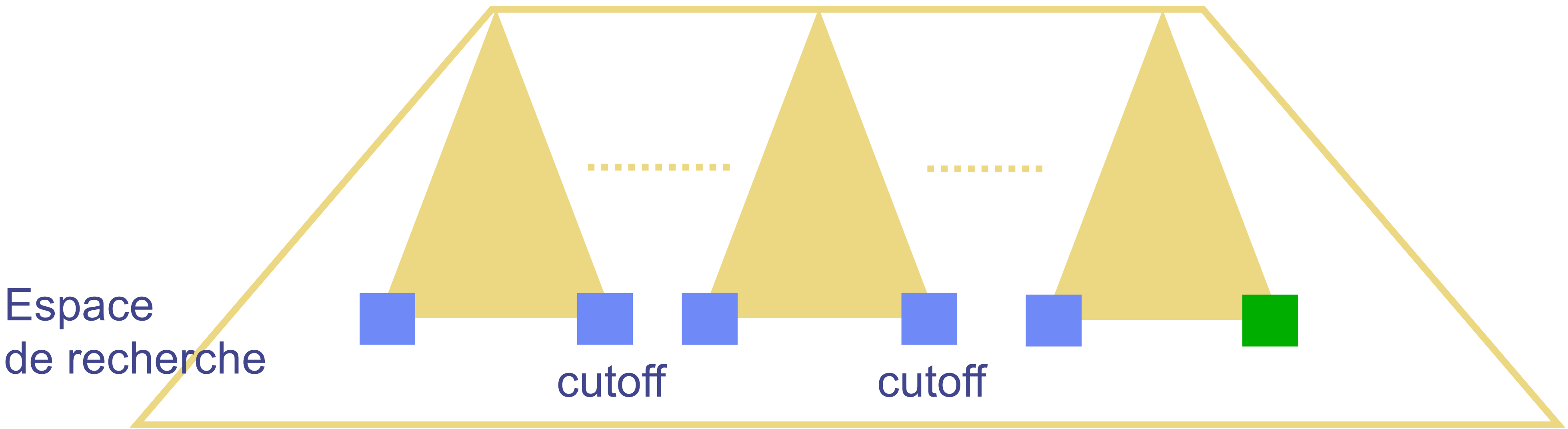
- Constatation:
  - pour un problème donnée, les temps d 'exécutions varient considérablement en fonction des heuristiques et/ou algorithmes

Problème de vérification de circuit
- heuristiques randomisées
- 10000 exécutions



Autres méthodes : « Randomization & restart »

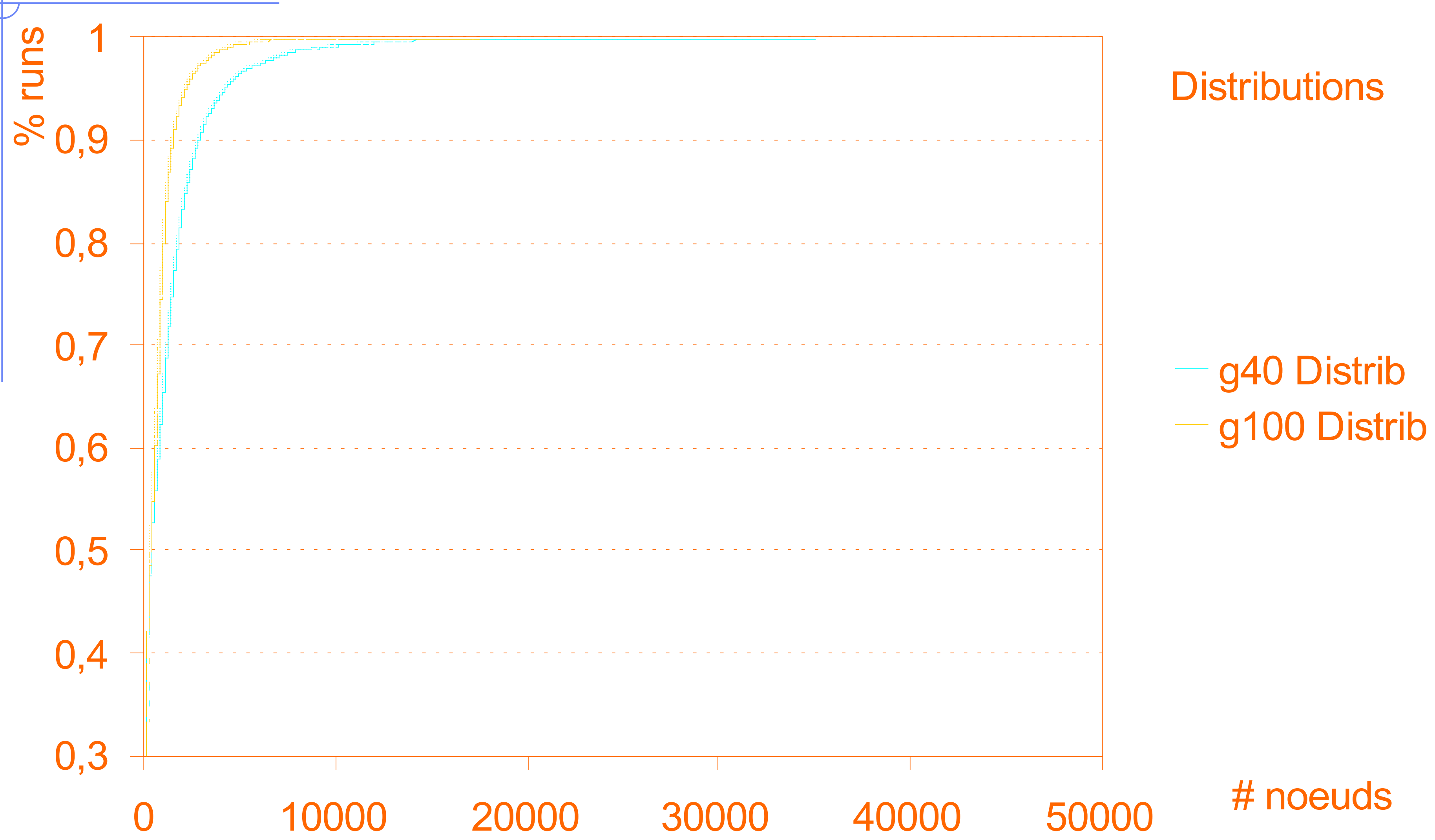
- Stratégie :
  - Randomizer l'heuristique de choix de variable
  - Limiter le nombre de nœuds à explorer à chaque essai "*backtrack cutoff value*"
  - reprendre la recherche à chaque fois que le nombre de nœuds>cutoff
    - utiliser une heuristique randomisé pour explorer différentes parties de l'espace de recherche



Autres méthodes : « Randomization & restart »

- Possibilité de rendre l'algorithme complet
  - augmenter la valeur "cutoff" après chaque essai
- On peut utiliser les technique de traitement des échecs
  - très utiles pour prouver l ' insatisfiabilité
- On peut utiliser différents algorithmes et/ou des configurations d 'algorithmes
  - Soit , exécuter *K* algorithms (ou une configurations d 'algorithmes)
    - de manière concurrentes, avec différents processeurs, ou
    - séquentiellement , avec un seul processeur
  - Soit, à chaque essai, choisir un algorithme différent

Autres méthodes : « Randomization & restart »



Méthodes mixtes

Méthodes mixtes

[Mazure-Saïs-Grégoire:95-97]

- Résolution de problèmes difficiles
  - Pas de méthode générale et efficace
  - Avantages /inconvénients ⇨ complémentaires

⇩

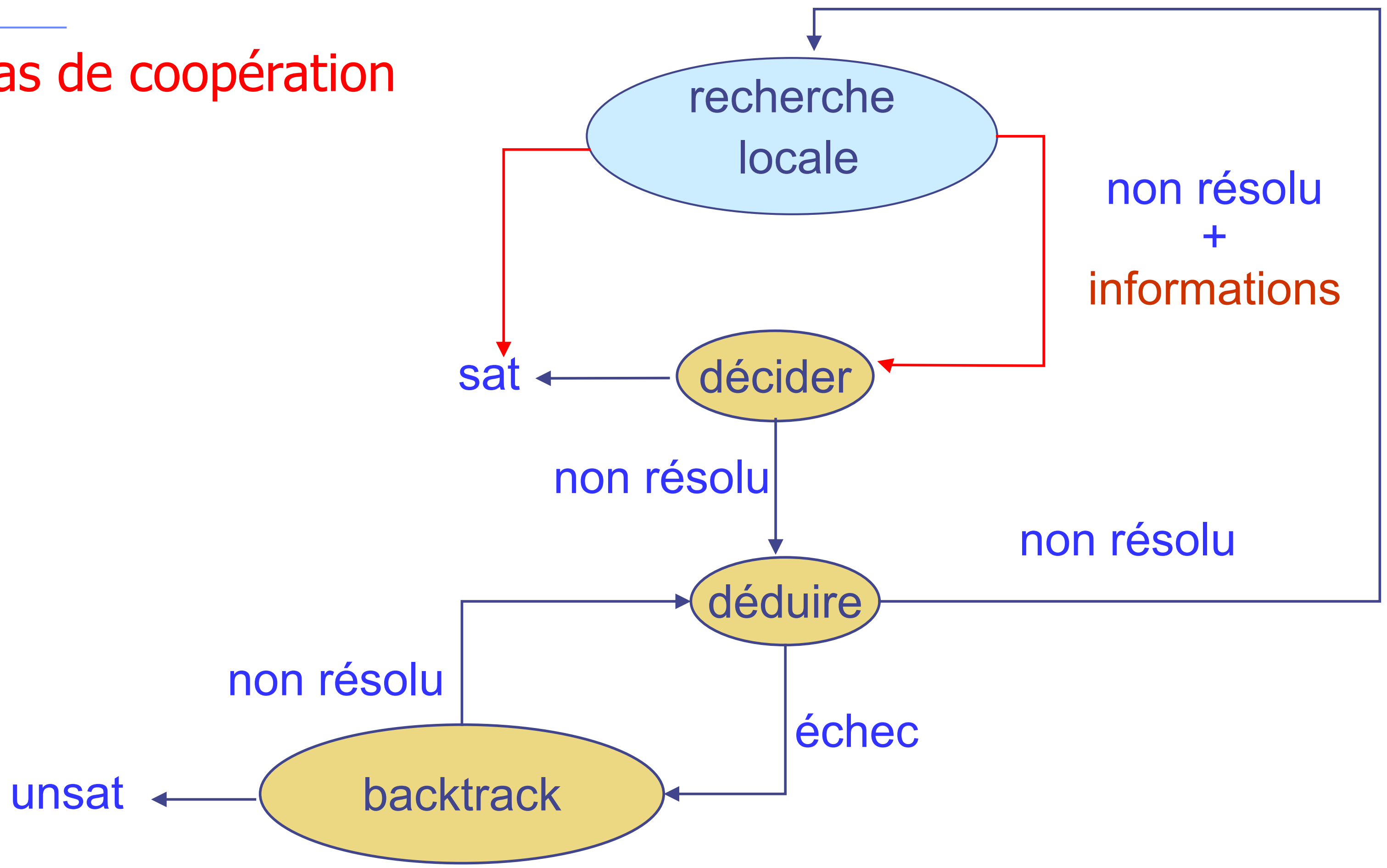
Hybridation de méthodes
(RO, PLC, …)

- Quelles méthodes ?
  - Recherche locale / énumération

- Comment ?
  - Avec ou sans coopération

Méthodes hybrides (mixtes)

- Schémas de coopération



- Autres travaux
  [Castell:97, Lobjois-Lemaître:97, ...]

Méthodes hybrides (mixtes)

- A chaque appel à l 'algorithme de recherche locale :
  - comptabiliser pour chaque clause le nombre de fois qu'elle est apparue fausse au cours de la recherche
  - Utiliser cette information pour améliorer les heuristique de choix de variable : e.g. choisir la variable ayant apparue le plus sauvent fausse au cours de la recherche locale.
- Sommaire :
  - Complétude des deux schémas proposés
  - Moyen de localiser l'inconsistance
  - Amélioration des techniques classiques sur de nombreux problèmes

Diagramme Binaire de Décision (BDD)

- Soit $x \rightarrow y, z$ un opérateur « if-then-else » (ite) défini :

$$x \rightarrow y, z = (x \wedge y) \vee (\neg x \wedge z)$$

↳ $x \rightarrow y, z$ est vrai si x et y sont vrai ou x est faux et z est vrai

Les opérateurs logique peuvent être exprimés par des «ite» et les deux constantes 0 et 1.

exemples :

- $\neg x$ par $x \rightarrow 0, 1$
- $x \rightarrow y$ par $x \rightarrow (y \rightarrow 1, 0), 1$
- $x \leftrightarrow y$ par $x \rightarrow (y \rightarrow 1, 0), (y \rightarrow 0, 1)$,
- $x \wedge y$ par $x \rightarrow (y \rightarrow 1, 0), 0$
- $x \vee y$ par $x \rightarrow 1, (y \rightarrow 1, 0)$
- ….

Remarque :

- les variables (sans négation) apparaissent uniquement dans la condition d'un ite. toute formule peut être représenté par les « ite », 0 et 1

↳ Nouvelle forme normale!

Diagramme Binaire de Décision (BDD)

- La forme normale INF « If-then-else Normal Form » est une expression booléenne construite avec uniquement des ite et les constantes 0 et 1, avec les tests effectuées uniquement sur les variables.
- Si φ est une formule booléenne et x une variable de φ alors,

$$\varphi = x \rightarrow \varphi[x|1], \varphi[x|0]$$

(théorème de Schanon)

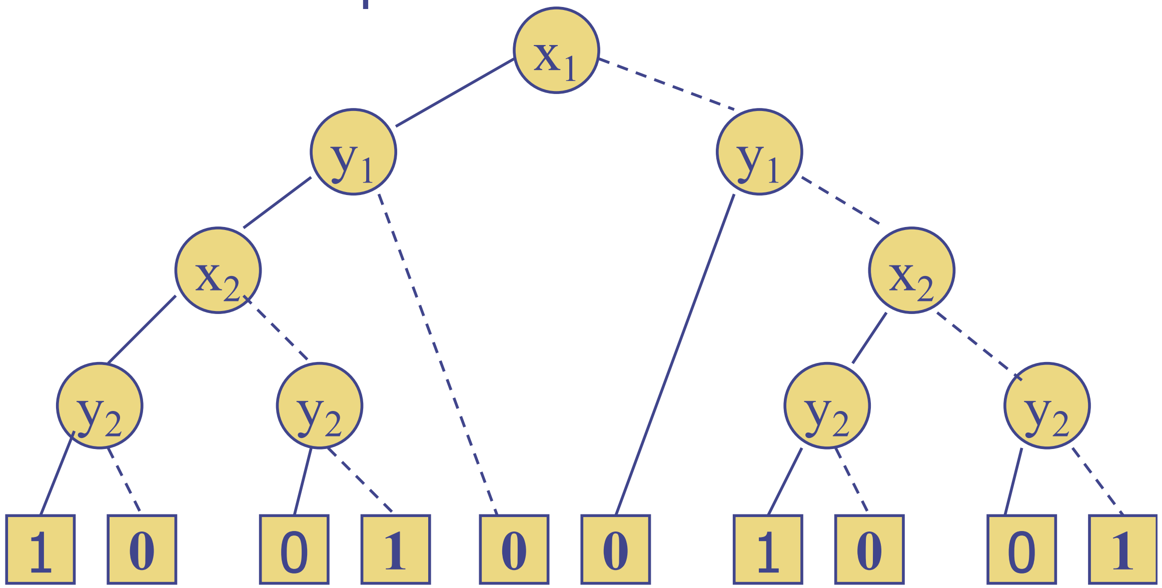
  ↳ En appliquant récursivement le théorème de Schanon on peut transformer toute formule booléenne sous forme INF

- toute formule booléenne est équivalente à une formule sous forme INF

Diagramme Binaire de Décision (BDD)

- Exemple 1 :
  soit une formule booléenne $\varphi = (x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2)$
  en appliquant récursivement le Th de Schanon sur les variables x_1, y_1, x_2, y_2 dans l'ordre on obtient l 'expression suivante:

$$\varphi = x_1 \rightarrow \varphi_1, \varphi_0$$
$$\varphi_0 = y_1 \rightarrow 0, \varphi_{00}$$
$$\varphi_1 = y_1 \rightarrow \varphi_{11}, 0$$
$$\varphi_{00} = x_2 \rightarrow \varphi_{001}, \varphi_{000}$$
$$\varphi_{11} = x_2 \rightarrow \varphi_{111}, \varphi_{110}$$
$$\varphi_{000} = y_2 \rightarrow 0, 1$$
$$\varphi_{001} = y_2 \rightarrow 1, 0$$
$$\varphi_{110} = y_2 \rightarrow 0, 1$$
$$\varphi_{111} = y_2 \rightarrow 1, 0$$



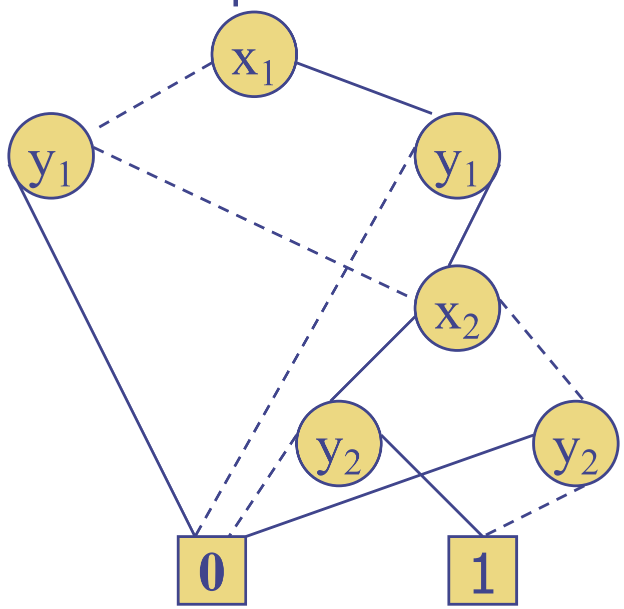
Un arbre de décision (AD)de φ

—— Étiqueté par 1

- - - - Étiqueté par 0

Diagramme Binaire de Décision (BDD)

- Exemple 1 (suite) :
  - chaque sous-formule peut être vue comme un neoud d 'un graphe.
  - Chaque nœud est soit terminal (1 et 0) soit non-terminal
  - un nœud non-terminal admet deux successeurs : la partie alors de l 'ite et la partie sinon de l 'ite



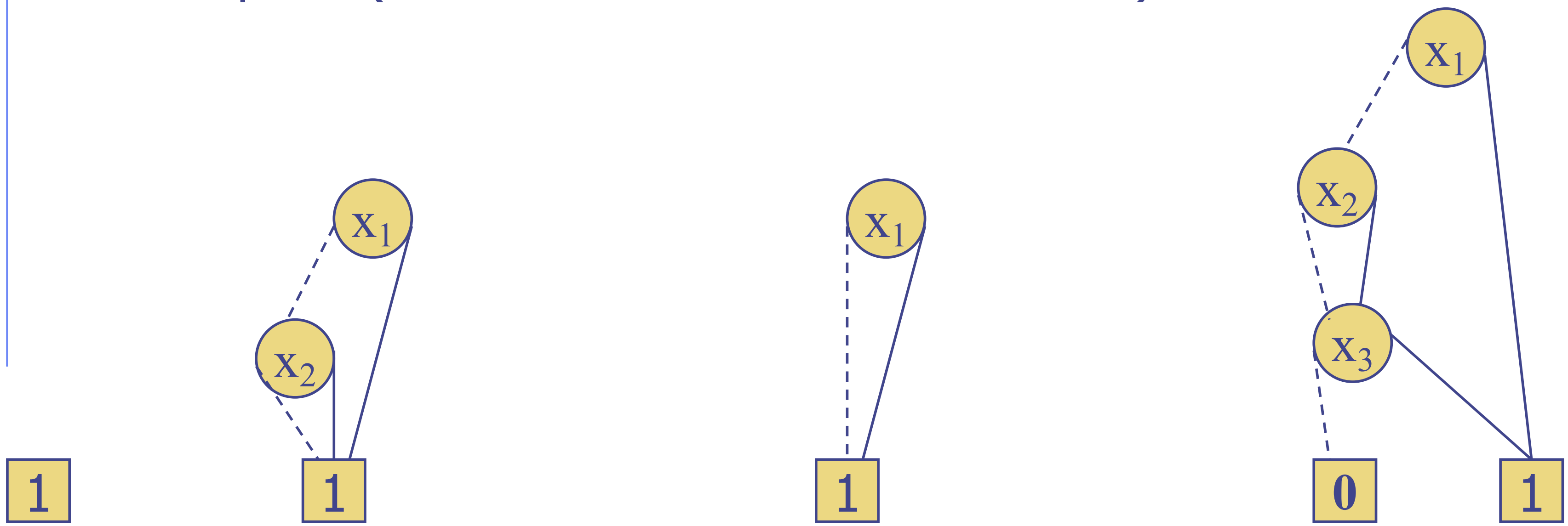
Nombre de nœuds réduit de 9 (AD) à 6 (BDD)

Un BDD de φ avec un ordre $x_1 < y_1 < x_2 < y_2$

↳ aussi OBDD « *Ordred Binary Decision Diagram* »

Diagramme Binaire de Décision (BDD)

- Exemple 2 (OBDD avec des tests redondants)



a) OBDD pour 1

b) OBDD pour 1, avec 2 tests redondants

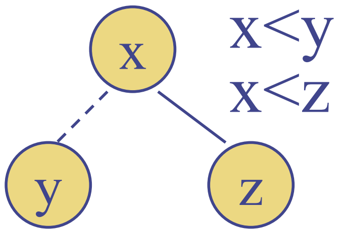
c) OBDD pour 1, avec 1 test redondant

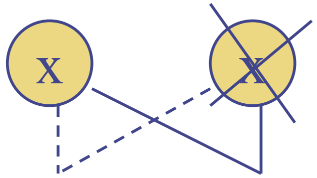
d) OBDD pour $x_1 \vee x_3$, avec 1 test redondant

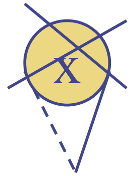
En éliminant les tests redondants dans un BDD on obtient un ROBDD (Reduced BDD)

Diagramme Binaire de Décision (BDD)

- ROBDD : ordre et réductions





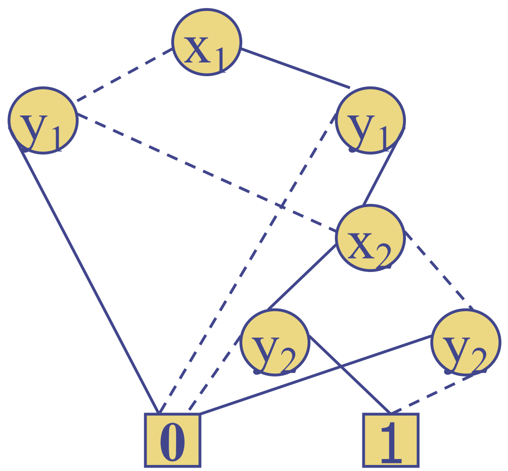


- Définitions :
  - un BDD est un graphe orienté acyclique (DAG) avec :
    - un ou deux nœuds terminaux (avec degré extérieur nul) étiquetés 0 ou 1, et
    - un ensemble de nœuds variables (non-terminaux) (avec un degré extérieur 2). Chaque nœud est étiqueté par une variable u, et chaque arc est étiqueté par 0 (ligne en pointillé) ou 1 (ligne pleine).
  - Un BDD est dit ordonné (OBDD) si sur chaque chemin du graphe l'ordre x1<x2 < … <xn est respecté. Un OBDD est réduit (ROBDD) si
    - (unicité) il ne contient pas deux neuds différents étiqueté avec la même variable et ayant les mêmes successeurs.
    - (non-redondant) les successeurs d'un neoud sont différents

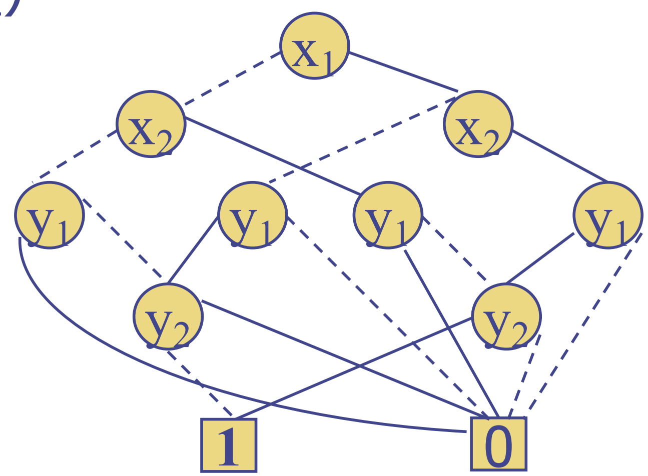
ROBDD

- Propriété (canonicité) : toute fonction booléenne est représenté par exactement un seul ROBDD.
  - une fois le ROBDD construit, :
    - on peut tester en temps constant si la formule booléenne est valide (ROBDD réduit au nœud 1), insatisfiable (ROBDD réduit au nœud 0), satisfiable sinon;
- ordre des variables -> taille du ROBDD

Exemple : $\varphi = (x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2)$



ROBDD (φ) : ordre $x_1 < y_1 < x_2 < y_2$



ROBDD (φ) : ordre $x_1 < x_2 < y_1 < y_2$

OBDD (structure implicite)

- obdd(T,{…})= 1
- obdd(⊥ ,{…})=0
- obdd(φ, {A1,A2,…,An})=

  if (A1)

  then obdd(φ[A1|1],{A1,A2,…,An})

  else obdd(φ[A1|0],{A1,A2,…,An})

OBDD (construction incrémentale)

- obdd(T,{…})= 1
- obdd(⊥ ,{…})=0
- obdd(φ_1 op φ_2 , {A1,A2,…,An})=

  obdd_merge(op,

  obdd_build(φ_1 , {A1,A2,…,An}),

  obdd_build(φ_2 , {A1,A2,…,An}),

  {A1,A2,…,An}

)

- op $\in$ {$\wedge$, $\vee$,$\leftrightarrow$,$\rightarrow$}

(RO)BDD : sommaire

- Opère sur des formules booléennes quelconques
- trouve tout les modèles
- factorisation des partie commune de l 'arbre de recherche (DAG)
- nécessite un ordre statique (fixé) à priori (choix critique!!)
- très efficace pour certaines classes de problèmes (e.g. circuits)
- nécessite un espace exponentiel dans le pire cas
- très utilisé par la communauté conception de matériel, ignoré par les logiciens, récemment introduite en IA