# Projet INF402 2020

Stéphane Devismes & Benjamin Wack

## 1 Démarche

Ce projet consiste à utiliser un SAT-solveur pour résoudre automatiquement des problèmes formalisés en clauses. Un SAT-solveur est un outil générique de détermination d'un modèle pour un ensemble de formules : votre travail ne consiste donc pas à appliquer des stratégies de résolution spécifiques à votre problème, mais à traduire le problème choisi dans le formalisme attendu par le SAT-solveur, et à laisser ce dernier déterminer une solution.

Vous devrez donc choisir un problème (grille de jeu logique type Sudoku) et vous en approprier les règles. Vous proposerez alors une modélisation d'une grille de ce problème à l'aide de variables booléennes uniquement, et vous traduirez les règles du jeu en formules logiques, plus spécifiquement sous forme normale conjonctive.

Une fois le problème modélisé et validé par votre chargé de TD, vous devrez programmer une interface logicielle **dans le langage de votre choix**, qui récupère des entrées définissant une instance de votre problème, et qui en sortie crée un fichier DIMACS codant la formule en forme normale conjonctive représentant cette instance.

Vous choisirez ensuite un SAT-solveur [1] pour traiter ces fichiers DIMACS.

Enfin, à partir de la trace produite par le SAT-solveur, vous afficherez alors la solution du problème de manière compréhensible.

À titre optionnel, selon l'avancement de votre groupe, vous avez la possibilité de transformer la modélisation de votre problème en clauses 3-SAT équivalentes (vous pourrez vous baser sur l'algorithme proposé dans l'exercice 51 p. 63 de votre polycopié de cours). Pour les groupes les plus avancés, vous pourrez également programmer votre propre SAT-solveur, ce qui sera facilité par la transformation en clauses 3-SAT.

**Note :** La note de projet est basée à parts équivalentes sur trois tâches :

- l'écriture d'un rapport clair, rigoureux et bien structuré ;
- la livraison d'un code source lisible, bien organisé et fourni avec des tests pertinents ;
- la participation active à une soutenance, qui inclura une démonstration du logiciel.

En particulier, tout candidat absent ou n'intervenant pas significativement durant la soutenance de son groupe s'expose à recevoir une note de 0 au projet.

1. Par exemple, MiniSat, Z3, Sattime, SAT4j, SATzilla, precosat, MXC, clasp, SApperloT, TNM, gNovelty2+, Rsat, Picosat, Minisat, Zchaff, Jerusat, Satzoo, Limmat, Berkmin, OKSolver, ManySAT 1.1, ... (`http://www.satcompetition.org/`).

# 2 Travail à réaliser

Pour ce projet les étudiants doivent former des groupes (de 3 personnes de préférence, 4 si besoin) et choisir un problème ou bien en proposer un à leur responsable de TD.

## 2.1 Rendus

Vous devrez produire :

1. Un rapport dans lequel :
   — vous présentez le problème choisi en français (règles, exemples, contraintes) ;
   — vous traduisez ce problème en logique (propositionnelle ou du premier ordre) ;
   — vous modélisez le problème choisi en forme normale conjonctive (produit de clauses) ;
   — vous détaillez les points critiques de l'implémentation de vos programmes ;
   — vous illustrez l'utilisation de vos programmes au moyen d'exemples bien choisis.
2. Une série de plusieurs programmes :
   — un programme qui récupère une entrée définissant une instance de votre problème, et qui crée un fichier DIMACS représentant cette instance ;
   Il est vivement conseillé (pour faciliter les tests) de récupérer les entrées dans un fichier [2] dont vous aurez défini le format. Facultativement, vous pouvez compléter ce type d'entrée par une saisie au clavier ou par une interface graphique.
   — le cas échéant, un programme qui lit un fichier au format DIMACS et renvoie un ensemble de clauses 3-SAT au format DIMACS ;
   — un programme qui, à partir de la trace produite par le SAT-solveur choisi, affichera la solution du problème de manière compréhensible ;
   — autant que possible, un programme qui enchaîne automatiquement ces étapes ainsi que l'appel au SAT-solveur.
3. Une série pertinente d'instances de votre problème, qui vous permettra de tester votre modélisation en clauses SAT.
4. Une soutenance où vous présenterez votre problème, sa modélisation, et où vous exécuterez vos programmes sur une machine personnelle ou de l'université.

## 2.2 Planning

— Le sujet du projet est en ligne dès le premier cours, une version synthétique est distribuée aux étudiants.
— Avant la fin de la 2$^{e}$ semaine de TD, constitution des groupes et choix des sujets.
— Livraison par email à votre enseignant de TD au plus tard le vendredi 6 mars 2020 du pré-rapport décrivant la partie modélisation du projet (5 pages maximum).
— Livraison au plus tard le vendredi 24 avril 2020 par email du rapport final et des codes sources des programmes.
— Dans la semaine du 27 au 30 avril, ou du 4 au 7 mai 2020, soutenance de 20 minutes par groupe, questions incluses (démonstration du travail réalisé en groupe).

2. Si vous programmez en JAVA, vous pouvez vous inspirer de l'exemple donné à la page `www-verimag.imag.fr/~devismes/JAVA/Fichier.java` pour la gestion de fichier.

# 3 Notes techniques

**SAT et $n$-SAT** Savoir si une formule de logique propositionnelle sous forme normale conjonctive (CNF) est satisfaisable est appelé "problème de satisfaisabilité" ou "problème SAT". Lorqu'une formule CNF ne contient que des clauses de $n$ littéraux on appelle ce problème : "problème $n$-SAT".

**Le format DIMACS :** Le format des fichiers d'entrées des SAT-solveurs est un standard international pour la représentation de formules en forme normale conjonctive. Un fichier en format DIMACS commence par une ligne qui spécifie qu'il s'agit d'une forme normale conjonctive, qui précise combien de variables la formule contient, et de combien de clauses disjonctives elle est constituée. Par exemple : `p cnf 5 3` indique que le fichier contient une formule en forme normale conjonctive, avec 5 variables et 3 clauses. Ensuite, le fichier est composé de plusieurs lignes, une par clause. Chaque ligne contient des entiers positifs et/ou négatifs, et se termine par 0. Un entier positif $i$ indique que la $i$-ème variable apparaît avec polarité positive dans la clause. Un entier négatif $-i$ indique que la $i$-ème variable apparaît avec polarité négative dans la clause. Par exemple, le fichier suivant au format DIMACS représente la formule :

$$(x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$$

```
c
c start with comments
c
c
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

# 4 Exemple de sujets possibles

Tous les jeux logiques consistant à remplir une grille en disposant de toutes les informations nécessaires sont modélisables. Par exemple, on pourra s'attaquer à :

- Facile :
  - N reines
  - Tout Noir Tout Blanc
- Moyen :
  - Usowan
  - Futoshiki
  - Dosun-Fuwari
  - Norinori
  - Coloration de graphes
- Difficile :
  - Light Up
  - Takuzu

— Hashiwokakero
— Tetravex
— Kurotto

— Autres sujets validés par votre chargé de TD...

D'autres pistes de sujets sont proposées sur la page
`https://wackb.gricad-pages.univ-grenoble-alpes.fr/inf402/`
avec en général des liens vers des applets de démonstration.

# 5 Exemple pour le problème des pigeons

Nous illustrons par un exemple simple le minimum que nous attendons des étudiants pour le projet de INF402.

## 5.1 Problème

Un colombophile possède $n$ nids et $p$ pigeons. Il souhaite que :
— chaque pigeon soit dans un nid,
— il y ait au plus un pigeon par nid.
Comment la logique peut-il l'aider ?

## 5.2 Modélisation en logique du premier ordre

Le prédicat $P(i, j)$ représente le fait que le pigeon $i$ est dans le nid $j$. Les contraintes du problème se modélisent comme suit :
— Chaque pigeon est dans un nid : $\forall i, \exists j, P(i, j)$.
— Il y a au plus un pigeon par nid : $\forall i, \forall k, i \neq k \Longrightarrow \forall j, \overline{P(i, j)} \vee \overline{P(k, j)}$.

## 5.3 Modélisation en forme normale conjonctive

La variable booléenne $x_{i,j}$ représente le fait que le pigeon $i$ est dans le nid $j$. Par exemple, pour un ensemble de pigeons $\{a, b, c\}$ et un ensemble de nids $\{1, 2, 3, 4\}$, nous devons donner au SAT-solveur les contraintes suivantes :

$$
\begin{aligned}
& (x_{a,1} \vee x_{a,2} \vee x_{a,3} \vee x_{a,4}) \\
\wedge\ & (x_{b,1} \vee x_{b,2} \vee x_{b,3} \vee x_{b,4}) \\
\wedge\ & (x_{c,1} \vee x_{c,2} \vee x_{c,3} \vee x_{c,4}) \\
\wedge\ & (\overline{x_{a,1}} \vee \overline{x_{b,1}}) \wedge (\overline{x_{a,1}} \vee \overline{x_{c,1}}) \wedge (\overline{x_{b,1}} \vee \overline{x_{c,1}}) \\
\wedge\ & (\overline{x_{a,2}} \vee \overline{x_{b,2}}) \wedge (\overline{x_{a,2}} \vee \overline{x_{c,2}}) \wedge (\overline{x_{b,2}} \vee \overline{x_{c,2}}) \\
\wedge\ & (\overline{x_{a,3}} \vee \overline{x_{b,3}}) \wedge (\overline{x_{a,3}} \vee \overline{x_{c,3}}) \wedge (\overline{x_{b,3}} \vee \overline{x_{c,3}}) \\
\wedge\ & (\overline{x_{a,4}} \vee \overline{x_{b,4}}) \wedge (\overline{x_{a,4}} \vee \overline{x_{c,4}}) \wedge (\overline{x_{b,4}} \vee \overline{x_{c,4}})
\end{aligned}
$$

## 5.4 Transformation en clauses 3-SAT et résolution par un SAT-solveur

Après avoir traduit la forme normale conjonctive en un fichier DIMACS, notre premier programme traduira ce fichier en un autre fichier DIMACS de clauses 3-SAT. Un SAT-solveur donnera ensuite une affectation à chaque variable satisfaisant les contraintes du

problème. Ainsi nous saurons comment placer les pigeons. Nous remarquons qu'il nous est facile de trouver une solution à ce problème alors que l'ordinateur doit résoudre un nombre de contraintes exponentielles.

# 6 Partie facultative : solveur SAT

Pour les groupes les plus avancés, vous avez la possibilité de **programmer** vous-même votre propre solveur SAT (dans le langage de votre choix) avec différentes stratégies de résolution.

Votre solveur, de type *WalkSat* (voir ci-dessous) devra lire en entrée l'ensemble de clauses à résoudre sous la forme d'un fichier DIMACS. Le but est ensuite de renvoyer une assignation modèle de toutes les clauses de l'ensemble si cela est possible. Notez que *WalkSat* est un solveur incomplet : si l'ensemble de clauses donné en entrée est contradictoire, il est incapable de le déterminer.

Le pseudo-code d'un algorithme de type *WalkSat* est donné ci-dessous.

```
 1: Choisir une assignation v uniformément au hasard
 2: i=0
 3: TANT QUE (v n'est pas modèle) et i < MAX_ITERATION FAIRE
 4:      Choisir au hasard C parmi les clauses C' telles que v(C') = faux
 5:      Tirer une valeur réelle x uniformément au hasard dans [0,1]
 6:      SI x <= P ALORS
 7:           Choisir uniformément au hasard une variable y de C
 8:      SINON
 9:           Choisir une variable y de C de manière déterministe
10:      FIN SI
11:      Inverser la valeur de v(y) dans l'assignation v
12:      i++
13: FIN TANT QUE
14: SI v est modèle ALORS
15:      Renvoyer v
16: SINON
17:      Renvoyer pas_de_decision
18: FIN SI
```

Le code donné ci-dessus est volontairement incomplet ! En effet, il ne précise pas les valeurs de MAX_ITERATION et P, ni la procédure de choix déterministe de la variable $y$ à la ligne 9.

Ce sera à vous de proposer une valeur raisonnable pour ces deux constantes, et de faire en sorte que la variable $y$ choisie soit la variable présente dans C qui va minimiser le nombre de clauses insatisfaites.

Vous pouvez enfin mener des expérimentations afin de trouver les meilleures valeurs possibles pour MAX_ITERATION et P ; vous pouvez également programmer d'autres méthodes de choix de $y$, par exemple :

— Choisir la variable présente dans C parmi celles les moins modifiées jusqu'ici.

— Pour chaque variable $z$, calculer $score(z)$ de la manière suivante :

  - si $v(z)$ = vrai alors compter le nombre de clauses où $z$ apparaît négativement moins le nombre de clauses où elle apparaît positivement,
  - sinon compter le nombre de clauses où $z$ apparaît positivement moins le nombre de clauses où elle apparaît négativement

  Ensuite choisir une variable de C de *score* maximum.
- Vous inspirer des heuristiques MOMS et JW données dans le poly.
- Mixer plusieurs des stratégies précédentes.
- etc.

Dans ce cas, il peut être judicieux de comparer entre elles les différentes procédures de choix que vous aurez implantées.

Notez que pour simplifier votre implémentation, vous pouvez supposer que l'ensemble de clauses reçu est constitué uniquement de clauses 3-SAT équivalent (voir début de la partie 2 de ce document).