# PROJET SPD 2019
# PARALLEL LOCAL SEARCH FOR SAT SOLVING

## Problème :

Le problème SAT (pour SATisfaisabilité d'une formule propositionnelle mise sous forme normale conjonctive) est un problème de décision qui consiste à déterminer si une formule CNF admet ou non un modèle. En logique propositionnel, une formule est sous forme normale conjonctive (CNF pour «Conjonctive Normal Form ») si c'est une conjonction de clauses. Une clause est une disjonction finie de littéraux de la forme ($l1 \vee l2 \vee \cdots \vee ln$). Un littéral est une variable propositionnelle ou bien sa négation. Associer une valeur à un ensemble de variables est une interprétation. Une interprétation est dite complète quand elle attribue des valeurs à toutes les variables. Le nombre d'interprétations possibles dépend du nombre de variables : avec X variables propositionnelles distinctes, il y a 2 puissance X interprétations possibles distinctes.

Soient la formule CNF $\Sigma = \{(a \vee b) \wedge (a \vee c)\}$ ainsi que deux interprétations associées $I1 = \{a, b, c\}$ et $I2 = \{\neg a, b, \neg c\}$. Nous avons $I1(\Sigma) = (\top \vee \top) \wedge (\top \vee \top) = \top \wedge \top = \top$ et $I2(\Sigma) = (\bot \vee \top) \wedge (\bot \vee \bot) = \top \wedge \bot = \bot$. Par conséquent, l'interprétation I1 est un modèle de Σ alors que I2 est un contre-modèle.

## La recherche locale :

Les approches incomplètes pour résoudre le problème SAT, sont en générale incapables de répondre à l'insatisfaisabilité d'une formule. Ces approches effectuent le plus souvent un parcours de l'espace de recherche non systématique pendant un temps donné. Par conséquent, puisque ce type d'approche ne garantit pas un parcours complet de l'espace de recherche de la formule, et cela, quelle que soit la quantité de temps laissée à la méthode. Il est donc impossible d'affirmer qu'il n'y a pas de solution. Ainsi, une fois le temps imparti écoulé, deux cas peuvent se présenter : soit un modèle est trouvé et la procédure stoppe son exécution et retourne le modèle obtenu, soit la méthode retourne qu'elle n'a pas trouvé de solution et dans ce cas il est impossible de conclure.

Il existe différents types d'approches incomplètes, les principales étant les techniques algorithmiques « génétiques » à base de population (Hao et Raphaël 1994), la « *survey propagation* » (Braunstein *et al.* 2005), la recherche à voisinage variable (Hansen *et al.* 2001), les algorithmes de colonies de fourmis (Dorigo et Stützle 2004), les algorithmes de parcours (Ow et Morton 1988) et la recherche locale. Dans cette section, nous présentons uniquement et globalement l'idée à l'origine de la recherche locale, celle-ci étant certainement la plus connue de toutes les méthodes incomplètes pour la résolution de SAT.

**Algorithme 2.1 :** Recherche Locale

**Données** : $\mathcal{F}$ une formule et $maxReparations$, le nombre maximum de réparations autorisées
**Résultat** : `vrai` si la formule $\mathcal{F}$ est satisfiable, `faux` s'il est impossible de conclure

1 **Début**
2 $\quad \mathcal{I} \longleftarrow$ une interprétation complète générée aléatoirement;
3 $\quad nbReparation = 0$;
4 $\quad$ **tant que** $(nbReparation < maxReparations)$ *et* $\mathcal{I}$ *n'est pas un modèle* **faire**
5 $\quad\quad$ **si** $\exists \mathcal{I}'$ *au voisinage de* $\mathcal{I}$ *telle que* $diff(\mathcal{F}, \mathcal{I}, \mathcal{I}') > 0$ **alors** $\mathcal{I} \longleftarrow \mathcal{I}'$ ;
6 $\quad\quad$ **sinon** $\mathcal{I} \longleftarrow$ interprétation choisie suivant un critère d'échappement ;
7 $\quad\quad nbReparation \longleftarrow nbReparation + 1$ ;
8 $\quad$ **si** $\mathcal{I}$ *est un modèle* **alors retourner** `vrai` ;
9 $\quad$ **retourner** `faux` ;

Concrètement, une méthode de recherche locale (algorithme 2.1) s'appuie sur une fonction de voisinage. Dans le cadre de SAT, le voisinage d'une interprétation sont toutes les interprétations qui ne différent que sur la valeur d'un seul littéral. Plus précisément, à partir d'une interprétation complète initiale, la méthode va se déplacer d'interprétation en interprétation voisine en permettant de réduire le nombre de clauses falsifiées. Dans le cas où il n'y a plus d'interprétation voisine permettant de diminuer le nombre de clauses falsifiées, et que l'interprétation courante n'est pas un modèle, alors un minimum local est atteint. Dans cette situation, il est nécessaire d'effectuer une perturbation dans l'interprétation courante. Cette perturbation est réalisée suivant une heuristique appelée critère d'échappement. Ainsi, c'est cette heuristique qui fait l'efficacité d'une méthode de recherche locale. Dans la littérature, de nombreux critères d'échappement ont été proposés (Mitchell *et al.* 1992, Selman *et al.* 1994). Cependant, en pratique les méthodes de recherche locale sont inefficaces sur les problèmes industriels. Dans ce cas les approches complètes restent la meilleure alternative.

## Partie 1 (12.5 points) : Parallélisme

L'objectif principal est de créer un solveur SAT de recherche locale parallèle utilisant à la fois la mémoire distribuée (MPI) et la mémoire partagé (Threads). Il s'agit d'une version parallèle concurrente (le premier trouvant un modèle arrête les autres) . Nous utiliserons pour ce projet un solveur SAT utilisant la recherche locale comme une boîte noire. En d'autres mots, aucune connaissance approfondie sur le problème SAT n'est requis pour réaliser ce projet. Vous devez créer trois versions parallèles :

- Une version multithreads (via les threads en C++11)
- Une version distribuée (via MPICH2)
- Une version hybride

Pour cela, vous avez à disposition sur mon site un solveur de recherche local nommé GSAT :

```
#include "GSAT.hpp"
using namespace std;

int main(int argc,char* argv[])
{
  //Sequential part: recuperate the problem and the options
  GSAT* gsat = new GSAT();
  gsat->setParameters(argc,argv);
  gsat->initialize();
  printf("c nbVariables: %d\n",gsat->getNbVariables());
  printf("c nbClauses: %d\n",gsat->getNbClauses());
  double startTime = gsat->realTime();

  //Parallel part: to do :)
  bool sat = false;
  while(!sat){
    sat=gsat->start();
    printf("c [iteration:%10d][file:%d][heuristic:%d]Satisfied clauses (begin:
%d)(end:%d)\n",
          gsat->getNbIterations(),
          gsat->getHeuristicFill(),
          gsat->getHeuristicSolve(),
          gsat->getNbSatisfiedClausesFill(),
          gsat->getNbSatisfiedClausesSolve());
  }
  //Sequential part: recuperate the solution and display the walltime
  printf("s %s\n",sat?"SATISFIABLE":"NOT FOUND");
  printf("c real time : %.4f seconds\n", gsat->realTime() - startTime);
  return sat;
}
```

A chaque itération (une boucle du “while”):

- Une interprétation complète est choisie via une heuristique de remplissage (getHeuristicFill)
- Ensuite, cette interprétation évolue (via une heuristique de résolution : `getHeuristicSolve()`) pendant la phase de résolution afin d’essayer de trouver un modèle en satisfaisant toutes les clauses.

## Partie 2 (7.5 points) : Exploiter le parallélisme

Cette partie est à programmer uniquement dans la version multithreads. La méthode start() permet de choisir les heuristiques via ses paramètres. Sans paramètres, des heuristiques sont choisie aléatoirement. Avec paramètres, elle est de la forme:

```
bool start(int fill,int heuristic);
```

Les heuristiques disponibles pour “fill” vont de 2 à 5 tandis que ceux disponible pour “heuristic” vont de 0 à 4.

### 2-1: Moyenne

Il s’agit ici de voir chaque heuristique comme un bandit manchot où le tirage du bras d’un bandit est le fait d'exécuter une heuristique. Une première version est alors de faire la moyenne des gains de chaque heuristique jouée.

- Initialisation : faire jouer tous les bandits (heuristiques) deux fois.
- Pour chaque bandit, faire la moyenne générale.
- A chaque prochaine itération, prendre les heuristiques qui sont les maximums des moyennes

Le gain est alors le nombre de clauses satisfaites. Ces mesures sont données pour les deux sortes d’heuristiques par les méthodes:

```
int getNbSatisfiedClausesFill();
int getNbSatisfiedClausesSolve();
```

Implémentation du calcul de la moyenne (incrémentalement):

$M(t)=(G(t1)+G(t2)+....+G(tk))/k$
$M(t+1)=M(t)+(1/(k+1))*(G(t(k+1)) - M(t))$

### 2-2 Epsilon Greedy

Prendre uniquement l’heuristique qui maximise les gains ne fait qu’exploiter une seule heuristique sans jamais explorer d’autre heuristique (Greedy). Dans cette partie, vous devez rajouter un pourcentage d’aléatoire (0.1,0.2,0.3,...,0.9,1)

d'exploration. Cette aléatoire est appelée Epsilon. Il doit donc avoir une option dans le programme (exemple: ./GSAT -epsilon=0.1 (Une fois sur 10, je choisis aléatoirement une heuristique, dans les autres cas, je prends celle maximisant les moyennes).

### 2-3 Moyenne glissante exponentielle

Faire la même chose, mais avec une moyenne exponentielle glissante.

**Définition 2.22** (Moyenne glissante exponentielle)**.** La **moyenne glissante exponentielle** (**EMA** pour *exponential moving average*) pour une série temporelle de nombre $\langle g_1, g_2, \dots g_n \rangle$, représentant l'évolution d'une quantité spécifique au cours du temps est calculé par :

$$EMA(\langle g_1, g_2, \dots g_n \rangle) = \alpha(1-\alpha)^{n-1} g_1 + \alpha(1-\alpha)^{n-2} g_2 + \cdots + \alpha(1-\alpha)^{n-i} g_i + \cdots + \alpha g_n$$

avec $0 < i < n$ et $\alpha$ un paramètre dit « de pas » tel que $0 < \alpha < 1$. Ce dernier permet de contrôler les poids qui différencient les données récentes des anciennes. La moyenne glissante exponentielle peut être calculée incrémentalement par :

$$EMA(\langle g_1, g_2, \dots g_n \rangle) = (1-\alpha) \times EMA(\langle g_1, g_2, \dots g_{n-1} \rangle) + \alpha g_n \text{ et } EMA(\langle\rangle) = 0$$

**Exemple 2.23.** Soient $\mathcal{S}_1 = \langle 1, 2, 3, 4 \rangle$ et $\mathcal{S}_2 = \langle 5, 4, 3, 2 \rangle$. Via une simple moyenne, nous avons $AVG(\mathcal{S}_1) = 2,5$ et $AVG(\mathcal{S}_2) = 3,5$. Par conséquent, $\mathcal{S}_2$ est le meilleur choix. En revanche, si nous calculons les moyennes glissantes exponentielles avec $\alpha = 0.5$. Nous avons $EMA(\mathcal{S}_1) = 3,0625$ et $EMA(\mathcal{S}_2) = 2,5625$. Dans cette situation, $\mathcal{S}_1$ est préférée à $\mathcal{S}_2$ car les données récentes ont un poids plus élevé que les anciennes.

L'idée est de faire une EMA pour chaque heuristique (toujour avec un epsilon greedy). Puis, à chaque itération, les deux heuristiques (une de remplissage et une de résolution) ayant les meilleures EMA sont choisies. Une fois l'itération terminée, les deux EMA associées doivent être mise à jour de cette manière. Soit une itération T, une heuristique H et un gain G. G(H,T) représente le gain d'une heuristique T pour une itération T.

EMA(H) = (1 − α) × EMA(H) + α *  G(H,T)

Deux mode sont demandés :

- Une version statique ou α est fixée à 0.5
- Une version dynamique ou α commence à 0.4 puis augmente petit à petit en fonction des itérations jusqu'à atteindre une valeur maximum de 0.6.

## Modalité:

Le projet est à réaliser seul ou en binôme. Deux séances de TPs seront réservées à la réalisation du projet. Une démonstration du logiciel aura lieu dans les salles TPs avant les examens. La date de rendu du projet est fixée après les examens au _____ (23h59). Le projet devra être délivré par mail à l'adresse suivante :

szczepanski.nicolas@gmail.com

sous la forme d'un fichier nommé "NOM1_PRENOM1_NOM2_PRENOM2.tar.gz".