# Planifier l'épreuve E5 à l'aide d'un solveur SAT

T Delacroix, H Sadighiyan

# Planifier l'épreuve E5 à l'aide d'un solveur SAT

T. Delacroix[1] H. Sadighiyan[2]

[1] IMT Atlantique - Dépt. LUSSI, Brest, France
[2] Lycée Jacquard, Paris, France

thomas.delacroix@imt-atlantique.fr

## Résumé

*Cet article propose une modélisation à l'aide d'un problème SAT d'un problème spécifique de définition d'un planning des passages d'étudiants en examen (épreuve E5 en BTS-SN) ainsi qu'un lien vers une implémentation en Python prête à l'emploi par les enseignants de la filière concernée permettant de résoudre le problème. La démarche est présentée de manière détaillée afin de permettre une adaptation aisée à d'autres contextes.*

## Mots Clef

emploi du temps, SAT, PicoSAT, BTS.

## Abstract

*This article addresses a specific timetabling problem for student exam schedules (E5 exam for technical university students enrolled in a French BTS-SN curriculum) by modeling it thanks to a SAT problem and providing a link to a ready-to-use Python implementation to solve this problem. The detailed presentation of the process makes it easily transposable to other situations.*

## Keywords

timetabling, SAT, PicoSAT, BTS.

# 1 Introduction

La gestion des emplois du temps est, sans aucun doute, l'un des grands casses-têtes administratifs des établissements scolaires et universitaires. En particulier, la gestion d'un planning pour les passages à l'oral par groupes des étudiants peut donner du sacré fil à retordre aux personnels administratifs et aux enseignants. Ainsi, il est traditionnel qu'en classe préparatoire, on demande à l'enseignant de mathématiques de préparer le colloscope (i.e. le planning des oraux) de la classe car il faut au moins être mathématicien pour intégrer l'ensemble des contraintes logiques à la solution du problème ! À l'ère de l'intelligence artificielle, les enseignants seraient en droit de se demander pourquoi les scientifiques n'ont toujours pas trouvé de solution pour les débarasser de ces problèmes.

Et pourtant, il s'agit tout simplement d'un gros malentendu. Car, bien au contraire, les chercheurs ont été particulièrement prolifiques dans le domaine de l'automatisation des emplois du temps (sûrement puisqu'ils sont confrontés à ce type de problèmes dans leur quotidien d'enseignants-chercheurs). À titre d'exemple, les états de l'art dans [17] et, plus récemment, dans [16] renvoient chacun à plus d'une centaine de publications dans ce domaine, dont les premières datent du début des années 60. Par ailleurs, la conférence internationale PATAT (Practice and Theory of Automated Timetabling `http://www.patatconference.org/`) réunit tous les deux ans de nombreux chercheurs autour de cette thématique. Alors, pourquoi tant de professeurs de mathématiques en classes préparatoires s'embêtent-ils tous les ans à refaire à la main leurs colloscopes ? Pourquoi les professeurs en BTS Systèmes numériques s'embêtent-ils tous les ans à refaire à la main les plannings des passages des étudiants pour les épreuves d'intervention sur système ?

Sans trop élaborer sur tous les obstacles à dépasser avant que le savoir d'une communauté scientifique soit mis en application au sein de la société de manière générale, il convient tout de même de faire quelques constats.

1. La diversité des contextes amène une diversité des solutions. Or ce n'est pas toujours évident de faire coller son propre problème à une méthodologie conçue pour en résoudre un autre.
2. La recherche scientifique se concentre sur des méthodes à la pointe, permettant de traiter les problèmes les plus complexes ou de les résoudre le plus rapidement possible. Mais celles-ci ne sont pas forcément les plus simples à implémenter ou accessibles à l'utilisateur.
3. L'utilisateur n'est pas forcément en capacité de transformer un algorithme en programme informatique, ni de se plonger dans un code de plusieurs milliers de lignes.

Ces différents constats doivent être pris en compte si l'on souhaite proposer un outil de résolution qui puisse être adopté localement, comme celui que l'on propose aux enseignants de BTS Systèmes numériques dans cet article.

Le premier constat incite à opter pour une méthode de résolution suffisamment flexible, où l'utilisateur peut définir chacune des contraintes de son problème. Cela signifie en

pratique que les contraintes doivent tout de même être implémentées directement par l'utilisateur. Le second constat oriente vers une méthode qui soit conceptuellement compréhensible pour l'utilisateur. Enfin, le dernier constat entraîne le choix d'une méthode dont les algorithmes principaux ont déjà été implémentés dans les langages de programmation courants et sont librement disponibles. Ainsi, les algorithmes de résolution peuvent demeurer une boîte noire pour l'utilisateur qui se limite à implémenter les contraintes de son problème.

Ces considérations nous ont amené à choisir, dans le cadre d'un problème de planning des passages des étudiants en épreuve E5 d'une classe de BTS Systèmes numériques, une méthode de résolution s'appuyant sur un solveur SAT [4, 18] implémentée en Python3. Ce type de solveur est largement assez performant pour apporter des solutions acceptables aux problèmes posés ici si l'on se limite aux effectifs d'une seule classe. Quant au choix du langage de programmation, il a été fait en cohérence avec l'importance croissante que ce langage a pris dans les lycées depuis l'introduction d'un enseignement d'algorithmique au programme de seconde en 2009 [7] et celle du langage Python au programme des classes préparatoires scientifiques en 2013 [6] et qui a conduit de fait à une plus grande maîtrise du langage au sein du corps enseignant.

### 1.1 Travaux connexes

Comme cela a été mentionné en introduction, les problèmes d'emplois du temps scolaires et universitaires ont fait l'objet de nombreux travaux scientifiques mais le nombre de travaux abordant ces problèmes sous la forme de problèmes SAT est relativement limité. Dans [17], l'auteur fait mention d'un possible axe de recherche à venir sur l'utilisation de techniques de résolution de problèmes SAT, à condition de modéliser le problème du *school timetabling* par un problème SAT. Toutefois, près de 10 années plus tard, il n'est fait aucune mention de telles démarches dans [16] qui est un état de l'art pourtant très complet. En contraste, on y retrouve de nombreux travaux implicant la résolution de problèmes CSP (Problèmes de satisfaction de contraintes), une approche proche mais bien distincte de la résolution de problèmes SAT [22].

Plus récemment, des articles ont, tout de même, abordé les problèmes de *timetabling* via le prisme des problèmes SAT [2] ou SMT [10], qui en est une généralisation. On peut également noter une approche SAT du *social golfer problem* qui n'est pas très éloigné des problèmes de gestion d'emplois du temps académiques [12, 21].

Enfin, il faut noter que le présent article se distingue de ces travaux en tant que l'on a fait le choix de ne pas inscrire la modélisation du problème considéré dans une classe de modélisations de problèmes pouvant être, à leur tour, modélisés par des problèmes SAT mais plutôt de passer directement à la modélisation en tant que problème SAT.

Plus largement, il existe différentes approches dont l'objectif général est apparenté à notre volonté de présenter une méthodologie qui permette une utilisation accessible d'outils existants pour la résolution de problèmes de programmation sous contraintes. On peut citer dans ce sens le développement du langage MiniZinc et de solutions logicielles complètes associées [15, 20]. Notre approche se distingue d'une telle approche car elle ne nécessite ni l'apprentissage d'un nouveau langage spécifique, ni l'installation de logiciel spécifique, mais est limitée, par contre, aux problèmes SAT. On peut également remarquer qu'une modélisation du présent problème via le langage MiniZinc ne nous a pas semblé être une tâche aisée.

## 2 Modélisation du problème

### 2.1 Présentation du contexte

Les étudiants de BTS SN-IR et de BTS SN-EC (Systèmes numériques - Informatique et réseaux ; Électronique et communications) doivent réaliser chaque année une épreuve intitulée E5-Intervention sur un système numérique et d'information [5]. En première année, pour chacun des étudiants, l'épreuve dure 12h découpées en plusieurs séquences. Chacune de ces séquences consiste pour les étudiants en une intervention en équipe sur un système numérique au cours d'une séance d'examen. Une équipe est composée d'un chef d'équipe et de techniciens. Chaque étudiant doit être mis en situation de chef d'équipe pendant 4 heures et en situation de technicien pendant 8 heures. Lors des séances d'examen, plusieurs équipes travaillent en parallèle, chacune sur son système, sous la surveillance d'un enseignant qui évalue les étudiants. Il y a généralement entre 4 et 6 types de systèmes différents proposés et comme chaque système nécessite du matériel spécifique, il n'est possible de faire travailler qu'un nombre très réduit d'équipes sur des systèmes de même type en parallèle.

Par exemple, au lycée Jacquard de Paris (19$^{e}$ arr.), les systèmes proposés sont DAMALISK (un réseau de téléphonie), WIFIBOT (un robot de surveillance), DMX (un système de gestion de projecteurs dans une salle de spectacle) et SIMRAD (un système de pilotage automatique de bateau). Le matériel disponible permet de faire tourner un système DAMALISK, un système SIMRAD, deux systèmes WIFIBOT et deux systèmes DMX en parallèle. Pour ces systèmes, une intervention requiert une équipe de trois étudiants (un chef d'équipe et deux techniciens) et se fait au cours d'une séance d'examen de 8 heures. Au cours de cette séance de 8 heures, chaque chef d'équipe travaille avec ses techniciens pendant 2 heures, ensuite les techniciens travaillent 4 heures en autonomie, puis le chef d'équipe travaille à nouveau avec les techniciens pendant 2 heures (ces dernières modalités peuvent varier d'un établissement à l'autre, mais cela n'a pas d'incidence sur le modèle proposé).

Ainsi, au cours d'une séance, un étudiant a : soit réalisé ses 4 heures en tant que chef d'équipe ; soit réalisé ses 8 heures en tant que technicien. Deux séances suffisent donc pour évaluer un étudiant. Toutefois, comme chaque intervention sur un système nécessite que deux techniciens y

consacrent 8 heures et que chaque étudiant doit passer au moins une fois en tant que chef d'équipe, il est demandé à chaque étudiant de passer deux fois en tant que technicien sur un système avec une évaluation sur un seul de ces passages. Ainsi, un étudiant doit travailler pendant trois séances, chacune correspondant à une équipe travaillant sur un système.

Les enseignants doivent donc proposer un planning pour des séances d'examen, en définissant les systèmes sur lesquels des équipes travailleront tout au long de cette séance et la composition de ces équipes, de manière à respecter les contraintes décrites précédemment. De plus, même si cela n'est pas imposé au niveau national, il paraît préférable qu'à chaque passage d'un étudiant, celui-ci travaille avec des étudiants différents et sur un système différent.

La tâche que le logiciel proposé dans la suite de l'article permet d'accomplir est la suivante :

*Étant donné un planning des séances d'examen et des systèmes à traiter au cours de chacune de ces séances, renvoyer, s'il en existe une, une répartition des étudiants en équipes d'intervention sur ces systèmes de manière à ce que :*

1. *chaque étudiant intervient trois fois exactement ;*
2. *chaque équipe est composée de trois étudiants exactement ;*
3. *un étudiant ne peut intervenir dans plus d'une équipe par séance ;*
4. *pour chaque étudiant, chaque intervention correspond à un système différent ;*
5. *chaque étudiant ne peut collaborer qu'une seule fois avec un autre étudiant ;*
6. *chaque équipe possède exactement un chef d'équipe ;*
7. *chaque étudiant doit passer une fois chef d'équipe.*

Le tableau 1 ci-contre donne un planning des systèmes par séances d'examen, et la répartition qui a été déterminée par le logiciel en fonction de ce planning. Chaque ligne correspond à un étudiant et chaque colonne à une séance. On indique qu'un étudiant est intervenu en tant que technicien ou en tant que chef d'équipe par les abréviations T et CE respectivement.

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| DMX | 2 | 2 | 1 | 1 |
| WIFIBOT (WIF) | 2 | 2 | 1 | 1 |
| DAMALISK (DAM) | 1 | 1 | 1 | 1 |
| SIMRAD (SIM) | 1 | 1 | 1 | 1 |

$\downarrow\downarrow\downarrow\downarrow$

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $E_1$ | T-WIF#2 | T-SIM#2 | CE-DMX#5 | |
| $E_2$ | CE-DAM#1 | T-WIF#4 | | T-SIM#4 |
| $E_3$ | T-WIF#1 | CE-DMX#4 | | T-DAM#4 |
| $E_4$ | CE-DMX#2 | T-DAM#2 | | T-WIF#6 |
| $E_5$ | CE-WIF#2 | T-DMX#4 | T-SIM#3 | |
| $E_6$ | T-DMX#2 | T-WIF#3 | | CE-SIM#4 |
| $E_7$ | | T-DMX#4 | CE-WIF#5 | T-SIM#4 |
| $E_8$ | | T-DMX#3 | CE-SIM#3 | T-DAM#4 |
| $E_9$ | CE-DMX#1 | T-WIF#3 | T-SIM#3 | |
| $E_{10}$ | T-DMX#2 | | T-WIF#5 | CE-DAM#4 |
| $E_{11}$ | CE-SIM#1 | T-DAM#2 | | T-DMX#6 |
| $E_{12}$ | T-SIM#1 | CE-DMX#3 | | T-WIF#6 |
| $E_{13}$ | T-DAM#1 | CE-WIF#3 | T-DMX#5 | |
| $E_{14}$ | T-DMX#1 | CE-DAM#2 | T-WIF#5 | |
| $E_{15}$ | T-DAM#1 | T-SIM#2 | | CE-WIF#6 |
| $E_{16}$ | T-SIM#1 | CE-WIF#4 | T-DMX#5 | |
| $E_{17}$ | T-WIF#1 | CE-SIM#2 | | T-DMX#6 |
| $E_{18}$ | T-DMX#1 | T-WIF#4 | CE-DAM#3 | |
| $E_{19}$ | CE-WIF#1 | T-DMX#3 | T-DAM#3 | |
| $E_{20}$ | T-WIF#2 | | T-DAM#3 | CE-DMX#6 |

TABLE 1 – Un planning des systèmes par séances et une répartition des étudiants admissible renvoyée par le logiciel

## 2.2 Résolution SAT

Le logiciel est construit autour d'une méthode de résolution d'un problème SAT. Sans rentrer dans les détails, en voici le principe.

Étant donnée une expression propositionnelle, il s'agit pour le problème de satisfiabilité (SAT) d'attribuer des valeurs de vérité aux variables propositionnelles de manière à ce que l'expression propositionnelle soit vraie, lorsque c'est possible, ou d'indiquer qu'il n'existe pas de telle solution sinon [13].

En pratique, les solveurs SAT sont généralement de type CNF-SAT (mis à part quelques rares exemples tel que [14]), ce qui signifie que l'expression propositionnelle doit être renseignée sous sa forme normale conjonctive (CNF). En particulier, c'est le cas pour le solveur PicoSAT sur lequel s'appuie la libraire pycosat utilisée dans le logiciel [4, 18].

Les solveurs SAT sont aussi généralement équipés d'un *timeout* qui leur permet de stopper la recherche s'ils n'ont pas pu aboutir à une solution au problème dans un délai raisonnable. C'est le cas de PicoSAT qui permet de fixer une limite au nombre de propagations unitaires qui peuvent être réalisées lors de la recherche (le nombre de propagations unitaires réalisées étant quasi-linéaire au temps d'exécution).

## 2.3 Base de la modélisation SAT

On commence par poser les bases de la modélisation, puis on modélise chacune des contraintes présentées précédemment.

On remarque que, comme chaque étudiant doit intervenir sur exactement trois systèmes et qu'il y a exactement trois étudiants par équipe, le nombre d'équipes doit être égal au nombre d'étudiants. On note $n$ ce nombre, $E_1, ..., E_n$ les étudiants et $A_1, ..., A_n$ les équipes.

Par ailleurs, chaque équipe travaille sur un système d'un certain type au cours d'une séance donnée. On note $t$ le nombre de types de systèmes différents et $s$ le nombre de séances sur lesquelles ils sont répartis. On fixe une partition $T_1 \sqcup ... \sqcup T_t$ de $[\![1, n]\!]$ de manière à ce que $A_j$ est une équipe travaillant sur un système de type $l$ si et seulement

si $j \in T_l$. De même, on fixe une partition $S_1 \sqcup ... \sqcup S_s$ de $[\![1, n]\!]$ de manière à ce que $A_j$ intervient lors de la séance $l$ si et seulement si $j \in S_l$. Pour faire cela, on part du planning des systèmes par séances que l'on représente sous la forme d'une matrice $P$ de taille $(t, s)$ et l'on attribue les indices de chacune des équipes en parcourant la matrice du haut vers le bas puis de droite à gauche. Dans le cas du planning des systèmes par séances donné en exemple précédemment, cela donne la matrice et les partitions ci-après.

$$\begin{array}{ll} T_1 = \{1, 2, 7, 8, 13, 17\} & S_1 = \{1, 2, 3, 4, 5, 6\} \\ T_2 = \{3, 4, 9, 10, 14, 18\} & S_2 = \{7, 8, 9, 10, 11, 12\} \\ T_3 = \{5, 11, 15, 19\} & S_3 = \{13, 14, 15, 16\} \\ T_4 = \{6, 12, 16, 20\} & S_4 = \{17, 18, 19, 20\} \end{array}$$

TABLE 2 – Partitions en types de systèmes et en séances définies pour l'exemple de la Table 1

Enfin, on modélise le fait qu'un étudiant $E_i$ participe ou non à une équipe $A_j$ par la variable propositionnelle $X_{i,j}$ et le fait qu'il en soit le chef d'équipe par la variable propositionnelle $Y_{i,j}$.

## 2.4 Modélisations des contraintes

On reprend ici une par une les contraintes listées à la section 2.1 en les exprimant en tant que contraintes de cardinalité lorsqu'il y a lieu. La contrainte de cardinalité $\#k(x_1, ..., x_n)$ signifie que $N \# k$ où $N$ est le nombre de variables évaluées à vrai parmi $x_1, ..., x_n$, $\#$ est l'un des symboles $\leq, =, \geq$ et $k$ un entier de $[\![0, n]\!]$.

**Contrainte (1) : trois équipes par étudiant.**

$$\forall i \in [\![1, n]\!], = 3(X_{i,1}, X_{i,2}, ..., X_{i,n}).$$

**Contrainte (2) : trois étudiants par équipe.**

$$\forall j \in [\![1, n]\!], = 3(X_{1,j}, X_{2,j}, ..., X_{n,j}).$$

**Contrainte (3) : non dédoublement des étudiants.**

$$\forall i \in [\![1, n]\!], \forall k \in [\![1, s]\!], \forall \mathbf{j} \in \mathcal{C}^2_{S_k}, \leq 1(X_{i,j_1}, X_{i,j_2})$$

où $\mathcal{C}^2_{S_k}$ désigne l'ensemble des combinaisons de deux éléments de $S_k$.

**Contrainte (4) : des systèmes différents.**

$$\forall i \in [\![1, n]\!], \forall k \in [\![1, t]\!], \forall \mathbf{j} \in \mathcal{C}^2_{T_k}, \leq 1(X_{i,j_1}, X_{i,j_2}).$$

**Contrainte (5) : des partenaires différents.**

$$\forall \mathbf{i} \in \mathcal{C}^2_n, \forall \mathbf{j} \in \mathcal{C}^2_n, \leq 3(X_{i_1,j_1}, X_{i_1,j_2}, X_{i_2,j_1}, X_{i_2,j_2}).$$

**Contrainte (6) : un chef d'équipe par équipe.**

$$\forall j \in [\![1, n]\!], = 1(Y_{1,j}, Y_{2,j}, ..., Y_{n,j}).$$

**Contrainte (7) : tout étudiant doit passer chef d'équipe.**

$$\forall i \in [\![1, n]\!], = 1(Y_{i,1}, Y_{i,2}, ..., Y_{i,n}).$$

**Contrainte (8).** On peut remarquer qu'un étudiant ne peut pas passer chef d'une équipe sans être dans cette équipe. On a donc une contrainte supplémentaire.

$$\forall i \in [\![1, n]\!], \forall j \in [\![1, n]\!], Y_{i,j} \implies X_{i,j}.$$

**Contrainte assouplie (4').** Comme indiqué dans la présentation du contexte, les contraintes qui ont été considérées ne sont pas toutes imposées au niveau national et on peut essayer de les assouplir pour obtenir des résultats plus facilement. On peut considérer seulement qu'un étudiant ne doit pas intervenir plus de deux fois (au lieu d'une) sur un même système.

$$\forall i \in [\![1, n]\!], \forall k \in [\![1, t]\!], \forall \mathbf{j} \in \mathcal{C}^3_{T_k}, \leq 2(X_{i,j_1}, X_{i,j_2}, X_{i,j_3}).$$

## 2.5 Encodage CNF performant des contraintes de cardinalité

Hormi, la contrainte (8), toutes les contraintes ici correspondent à des contraintes de cardinalité de type $\#k(x_1, ..., x_n)$. Ces contraintes peuvent être exprimées directement sous forme CNF via une approche dite naïve mais cela pose rapidement des problèmes de complexité dès que $n$ augmente car le nombre de clauses de ces contraintes est prohibitif. Il existe dans la littérature de nombreux encodages plus performants de ce type de contraintes qui permettent d'obtenir une forme CNF équisatisfiable comportant un nombre très réduit de clauses en rajoutant un nombre raisonnable de variables supplémentaires [1, 3, 11, 19]. Le logiciel proposé utilise de tels encodages en suivant le protocole défini dans [8].

## 2.6 Choix du planning des systèmes par séances

Jusqu'à présent dans cette section, il n'a été question que de la recherche d'une solution à un problème SAT défini par la matrice $P$ du planning des systèmes par séances, cette matrice étant définie a priori par l'enseignant. La question du choix de ce planning est également une question valable et il peut être envisagé d'automatiser ce processus.

En effet, on commence par faire la remarque suivante : si le problème SAT défini par une matrice $P$ admet une solution alors la matrice vérifie nécessairement les trois contraintes suivantes :

- la somme des coefficients de $P$ vaut $n$ ;
- pour chacune des colonnes de $P$, la somme des coefficients de cette colonne vaut au plus $n/3$ (sinon il faudrait qu'un étudiant soit présent dans deux équipes au cours de la même session, ce qui est proscrit) ;
- pour chacune des lignes de $P$, la somme des coefficients de cette ligne vaut au plus $n/3$ (sinon il faudrait qu'un étudiant interviennent sur deux systèmes du même type, ce qui est également proscrit).

Par ailleurs, on rappelle que, pour chaque type de système, il y a une borne sur le nombre de systèmes de ce type pouvant fonctionner simultanément. On note $R = (R_1, ..., R_t)$ ces restrictions.

Enfin le nombre de systèmes est fixé et on peut fixer le nombre de séances possibles (ou une borne maximale si l'on préfère).

Ainsi, en ne considérant que les matrices qui vérifient l'ensemble de ces contraintes, on limite sérieusement les possibilités. Pour le cas présenté en exemple précédemment, c'est-à-dire 20 étudiants, travaillant pendant au plus 4 séances sur les 4 systèmes du lycée Jacquard avec les restrictions associées sur le fonctionnement en parallèle de ces systèmes (i.e. $R = (2, 2, 1, 1)$), il existe exactement 100 matrices satisfaisant ces contraintes. Mathématiquement, c'est l'ensemble $\mathcal{P}(n, s, t, R)$ des matrices $P \in \text{Mat}_{t,s}(\mathbb{N})$ telles que :

$$\begin{cases} \sum_{i=1}^{t} \sum_{j=1}^{s} P_{i,j} = n, \\ \forall j \in [\![1, s]\!],\ \sum_{i=1}^{s} P_{i,j} \leq \frac{n}{3}, \\ \forall i \in [\![1, t]\!],\ \sum_{j=1}^{t} P_{i,j} \leq \frac{n}{3}, \\ \forall i \in [\![1, t]\!],\ \forall j \in [\![1, s]\!],\ P_{i,j} \leq R_i. \end{cases} \quad (1)$$

Un tel ensemble peut facilement être généré grâce à un algorithme récursif [9]. Comme cet ensemble est fini et calculable en un temps très raisonnable, il est envisageable de l'ordonner totalement par ordre de préférence selon un certain nombre de critères définis par les enseignants (comme regrouper les épreuves sur un minimum de séances ou proposer un minimum de types de systèmes différents). On peut ensuite parcourir cet ensemble ordonné de matrices, jusqu'à la première dont le problème SAT associé possède une solution.

Toutefois, dans la pratique cela n'est pas forcément la meilleure option et ceci pour deux raisons. D'abord, définir une fonction de préférence n'est pas immédiat et les critères de préférence sont tout à fait fluctuants d'un enseignant à l'autre. Ensuite, et c'est une découverte que nous avons réalisée en faisant différents tests sur le logiciel, pour toutes les valeurs testées, le problème SAT associé à chacune des matrices de $\mathcal{P}(n, s, t, R)$ admet une solution. Cette dernière remarque nous amène à poser la conjecture que les contraintes nécessaires énoncées dans cette section seraient en fait suffisantes pour que le problème SAT associée à une matrice $P$ possède une solution. Si tel est le cas, il ne sert à rien d'ordonner totalement l'ensemble de ces matrices, il suffit simplement d'en choisir une (la meilleure).

On remarque par ailleurs que les considérations faites dans cette sous-section ne sont valables que si l'on considère les contraintes dans leur version première et ne tiennent plus si l'on considère la contrainte assouplie (4').

| $n$ | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|
| $s_{min}$ | 3 | 4 | 4 | 4 | 4 | 5 | 4 |
| $\lvert\mathcal{P}\rvert$ | 1 | 1120 | 100 | 208 | 16 | 11700 | 1 |

TABLE 3 – Seuils minimaux $s_{min}$ et cardinaux des ensembles de matrices potentielles en fonction de $n$

# 3 Implémentation et résultats

## 3.1 Implémentation Python

L'implémentation proposée transpose directement ce qui a été décrit précédemment et permet de créer un fichier CSV contenant la solution si elle a été trouvée. À ce stade, la solution est calculée uniquement à partir d'une matrice $P$ rentrée par l'utilisateur. Nous intégrerons également, à terme, une option qui permettra au logiciel de fournir lui-même une telle matrice. Le code source en Python3 est mis à disposition librement sous licence GNU [9].

## 3.2 Le cas d'une classe : $18 \leq n \leq 24$

Étant donné qu'une classe de BTS Systèmes numériques a un effectif situé entre 18 et 24 étudiants, nous avons déterminé pour chacun des effectifs possibles toutes les matrices vérifiant les conditions nécessaires définies au niveau de la sous-section 2.6 (en respectant les contraintes spécifiques sur les systèmes du lycée Jacquard de Paris) de taille $(t, s)$ où $s$ a été choisi comme étant le seuil minimum $s_{min}$ à partir duquel il existe au moins une telle matrice.
Après vérification sur l'ensemble de ces matrices, chacune correspond à un problème SAT admettant une solution d'où notre conjecture que les conditions nécessaires de la sous-section 2.6 sont suffisantes. Nous n'avons toutefois pas cherché à démontrer mathématiquement cette conjecture.
Les deux tableaux suivants permettent de synthétiser les résultats obtenus.
Nous avons également cherché à voir s'il était possible de réduire le nombre de séances en assouplissant les contraintes, puisqu'il peut être compliqué pour les enseignants d'organiser de nombreuses séances. Étant donné qu'il ne peut y avoir plus de 6 systèmes par séance, il est clair que l'effectif maximal pour une valeur fixe de $s$ ne peut être supérieur à $6s$. On peut donc en déduire qu'un assouplissement des contraintes ne permettra pas de diminuer le nombre de séances, sauf éventuellement pour $n = 23$. Nous avons donc effectué des tests en remplaçant la contrainte (4) par sa version assouplie (4') et vérifié que, dans ce cas, la matrice $P$ suivante permettait d'obtenir une solution au problème SAT assoupli.

$$P = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

On précise également que le temps d'exécution est très satisfaisant par rapport à l'utilisation qui est attendue du logiciel. En effet, pour chacun des différents problèmes SAT

considérés, le temps d'obtention d'une solution est inférieur à la seconde, sachant que les calculs ont été réalisés sur un ordinateur portable bas de gamme au regard des standards actuels (processeur i5-2540M CPU @ 2.60GHz $\times$ 4 et mémoire vive de 3.7Go).

### 3.3 Gérer la complexité pour des plus grandes valeurs de $n$

Une analyse de l'algorithme proposé permet d'approcher le nombre de variables de la modélisation par $14n^2$ et le nombre de clauses par $\frac{1}{4}n^4 - \frac{1}{2}n^3 + \frac{173}{4}n^2 - 20n$. Ces quantités restent raisonnables pour un solveur SAT opérant sur un ordinateur lambda si l'on considère les valeurs de $n$ de la sous-section précédente. Cependant, si l'on considère des valeurs plus importantes comme $n = 80$, cela devient plus compliqué car la mémoire vive de l'ordinateur pourra saturer simplement sur le stockage des contraintes (dans l'implémentation Python, une clause est un tableau de jusqu'à 4 entiers de 28 bytes chacun, l'ensemble des clauses étant lui-même stocké dans un tableau). Même si, dans notre exemple particulier, il n'y a pas autant d'étudiants, ce nombre pourrait être largement dépassé si l'on souhaite organiser le déroulement d'épreuves semblables pour l'ensemble des étudiants d'une même académie.

Une solution consiste à découper le problème en sous-problèmes disjoints. Si l'on peut obtenir une solution à chacun de ces sous-problèmes, alors on peut les regrouper pour obtenir une solution au problème complet. Pour notre cas d'étude, il s'agit de considérer un premier groupe d'étudiant qui tourne sur un premier ensemble de séances, puis un deuxième groupe d'étudiant disjoint du premier qui tourne sur un nouvel ensemble de séances, également disjoint du premier, et ainsi de suite. Un tel découpage n'est pas toujours possible a priori. Toutefois, on peut montrer que, dans le cas précis qui est étudié ici (i.e. $4$ systèmes différents et $R = (2, 2, 1, 1)$), c'est non seulement faisable mais que cela permet également d'obtenir une solution avec un nombre de séances minimal dès que $n \geq 18$.

Pour montrer cela, on commence par remarquer que le nombre de séances est minoré par $\lceil n/6 \rceil + 1$ si $n \equiv -1 \mod 6$ ou par $\lceil n/6 \rceil$ sinon. En effet, les restrictions définies par $R$ limitent à 6 le nombre d'équipes qui peuvent intervenir au cours d'une même séance et, dans le cas où $n \equiv -1 \mod 6$, on doit nécessairement considérer une séance de plus de façon à ce que la somme des éléments d'une ligne de la matrice $P$ ne dépasse $n/3$. Par ailleurs, nous avons fait tourner le logiciel proposé pour toutes les valeurs de $n$ dans l'intervalle $[\![18, 35]\!]$ en considérant à chaque fois une matrice potentielle $P$ dont le nombre de colonnes (i.e. le nombre de séances) est égal à ce minorant. Dans tous les cas, nous obtenons une solution au problème SAT, ce qui démontre que ce minorant est un minimum pour toutes ces valeurs de $n$. Il suffit maintenant, pour tout entier $n \geq 18$, de le décomposer sous la forme $n = 18p + r$ avec $r \in [\![18, 35]\!]$ et $p \in \mathbb{N}$. Si l'on considère une solution avec un minimum de séances pour 18 étudiants et une solution avec un minimum de séances pour $r$ étudiants, alors il suffit de recoller $p$ fois la première solution, translatée à chaque fois sur un nouveau groupe de 18 étudiants et 18 équipes, puis une fois la deuxième solution, sur un dernier groupe de $r$ étudiants et $r$ équipes. On obtient alors bien une solution au problème pour $n$ étudiants, avec un nombre de séances égal au minorant décrit précédemment donc, a fortiori, minimal. La complexité pour le solveur SAT est alors constante, il n'est utilisé que deux fois sur des problèmes de taille limitée. Quant au découpage et au recollement, ils représentent une part négligeable de l'utilisation des ressources informatiques pour les valeurs de $n$ que l'on pourra être amené à considérer ici.

On rappelle que la démonstration qui précède ne porte que sur le cas où $R = (2, 2, 1, 1)$. Toutefois, la méthodologie du découpage, même si elle ne permet pas nécessairement d'obtenir une solution optimale, permettra généralement d'obtenir une solution alors même que les valeurs de $n$ sont trop importantes pour une approche directe. Ce processus n'a pas, à ce stade, été implémenté dans le code Python mis à disposition mais devrait y être intégré prochainement.

## 4 Conclusion

L'objectif premier de cet article est de proposer un logiciel permettant de répondre à un besoin précis des enseignants en BTS Systèmes numériques, qui puisse être utilisé de manière immédiate par un enseignant ayant un usage basique de Python, et permettant de servir de base pour une adaptation à d'autres problèmes par un utilisateur plus expérimenté.

L'objectif second de cet article est d'illustrer, par cet exemple, la relative facilité qu'il peut y avoir à modéliser un problème réel par un problème SAT, à implémenter cette modélisation et à obtenir, ainsi, une solution rapidement à ce problème. Il est vrai que, pour le problème que nous avons considéré, de nombreuses approches sont possibles. Si la conjecture énoncée à la sous-section 2.6 était avérée et qu'il en existait une preuve constructive, alors une approche mathématique permettrait peut-être de développer un algorithme plus performant. Mais cette démarche est incertaine et moins généralisable que l'approche présente. En contraste, il était ici quasi certain, dès l'énonciation du problème, qu'une approche via une modélisation SAT permettrait d'y apporter une réponse en un temps raisonnable. Cette certitude est bien évidemment liée aux faibles dimensions du problème, mais un grand nombre de problèmes ont de faibles dimensions. Le développement de telles approches pourrait aider à la résolution de nombreux problèmes qui prennent bien plus de temps qu'ils ne devraient, à commencer par la gestion administrative de nos enseignements.

*Les auteurs de cet article souhaitent remercier l'ensemble des étudiants en première année de BTS Systèmes numériques au lycée Jacquard de Paris qui ont servi de cobayes à cette expérience. Aucun n'étudiant n'a été maltraité durant ce processus.*

# Références


[1] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks : a theoretical and empirical study. *Constraints*, 16(2) :195–221, 2011.

[2] R. Asín Achá and R. Nieuwenhuis. Curriculum-based course timetabling with sat and maxsat. *Annals of Operations Research*, pages 1–21, 2012.

[3] O. Bailleux, Y. Boufkhad, and O. Roussel. New encodings of pseudo-boolean constraints into cnf. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 181–194. Springer, 2009.

[4] A. Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4 :75–97, 2008.

[5] BO. « systèmes numériques » option a « informatique et réseaux », option b « électronique et communications » : définition et conditions de délivrance. *Bulletin Officiel*, 47 du 19 décembre 2013, 2013.

[6] BO. Programmes des classes préparatoires aux grandes écoles. *Bulletin Officiel*, spécial du 30 mai 2013, 2013.

[7] DEGESCO. Ressources pour la classe de seconde - algorithmique -, 2009.

[8] T. Delacroix. Choisir un encodage cnf de contraintes de cardinalité performant pour sat. In *CNIA, Conférence Nationale en Intelligence Artificielle*, 2018.

[9] T. Delacroix. Sat pour e5. `https://github.com/Tdelacroix/SATpourE5`, 2018.

[10] E. Demirovic and N. Musliu. Solving high school timetabling with satisfiability modulo theories. pages 142–166, 2014.

[11] A. M. Frisch and P. A. Giannaros. Sat encodings of the at-most-k constraint. some old, some new, some fast, some slow. In *Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation*, 2010.

[12] I. Gent and I. Lynce. A sat encoding for the social golfer problem. In *IJCAIâĂŹ05 workshop on modelling and solving problems with constraints*, 2005.

[13] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. *Foundations of Artificial Intelligence*, 3 :89–134, 2008.

[14] R. Muhammad and P. J. Stuckey. A stochastic non-cnf sat solver. In *Pacific Rim International Conference on Artificial Intelligence*, pages 120–129. Springer, 2006.

[15] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. Minizinc : Towards a standard cp modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP'07, pages 529–543, Berlin, Heidelberg, 2007. Springer-Verlag.

[16] R. Qu, E. K. Burke, B. McCollum, L. T. Merlot, and S. Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of scheduling*, 12(1) :55–89, 2009.

[17] A. Schaerf. A survey of automated timetabling. *Artificial intelligence review*, 13(2) :87–127, 1999.

[18] I. Schnell. pycosat : bindings to picosat (a sat solver). `https://github.com/ContinuumIO/pycosat`, 2016.

[19] C. Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. *CP*, 3709 :827–831, 2005.

[20] P. J. Stuckey, T. Feydy, A. Schutt, G. Tack, and J. Fischer. The minizinc challenge 2008–2013. *AI Magazine*, 35(2) :55–60, 2014.

[21] M. Triska and N. Musliu. An improved sat formulation for the social golfer problem. *Annals of Operations Research*, 194(1) :427–438, 2012.

[22] T. Walsh. Sat v csp. *Principles and Practice of Constraint Programming–CP 2000*, pages 441–456, 2000.