Méthodes et modélisation pour l'optimisation

M1 informatique

2019–2020

INSTITUT D'ÉLECTRONIQUE
ET D'INFORMATIQUE
GASPARD-MONGE

Trouver des informations

- monge.univ-mlv.fr/~thapper/teaching/mmpo/
  - slides
  - feuilles de td et de tp
  - exercices travaillés
  - anciens examens
- Responsable de cours
  - Johan Thapper <thapper@u-pem.fr>

Organisation

- Cours 6 x 2h
  - Johan Thapper
- TD/TP 6 x 2h
  - Alfredo Hubard — Groupe ``Initiaux 1''
  - Anthony Labarre — Groupe ``Initiaux 2''
  - Johan Thapper — Groupe ``Apprentis''

Contrôle des connaissances

- 1 examen (janvier)
  - sur les notions abordées aux cours, TD et TP
  - 1 feuille recto-verso manuscrite autorisée
  - calculatrice, machine, smartphone, smartwatch interdits
- Pour réussir à l'examen
  - présence (pas seulement physique) au cours
  - travail aux TD et TP indispensable

Contrôle des connaissances

- Devoir x 2
  - à rendre **en personne sur la feuille distribuée**
  - vous avez droit de travailler ensemble mais **chacun rédige sa propre solution**
  - accorde 0,5 point sur l'examen / devoir complété

Acquis supposés

- Algèbre linéaire
  - indépendance linéaire
  - élimination de Gauss

$$2x_1 - 2x_2 + x_3 = 3$$
$$x_1 + 3x_2 - x_3 = 1$$
$$3x_1 - x_2 - x_3 = 2$$

- Vocabulaire de la logique propositionnelle
  - variable Booléenne, forme normale conjonctive (CNF), affectation
- Vocabulaire de la théorie des graphes
- Complexité
  - ordres de grandeur — temps polynomial vs exponentiel
  - problème NP-difficile — SAT, coloration de graphe, …

Modélisation et résolution d'un problème

Une entreprise veut installer la fibre optique entre ses bâtiment **A**, **B**, **C**, **D** et **E**. Le coût de chaque lien potentiel (point à point) est donné dans le tableau suivant :

	A	**B**	**C**	**D**	**E**
A		10	15	10	
B	10			7	18
C	15			15	
D	10	7	15		20
E		18		20	

Minimiser le coût total de l'installation.

Supposition : il suffit d'avoir un chemin entre chaque paire de bâtiments.

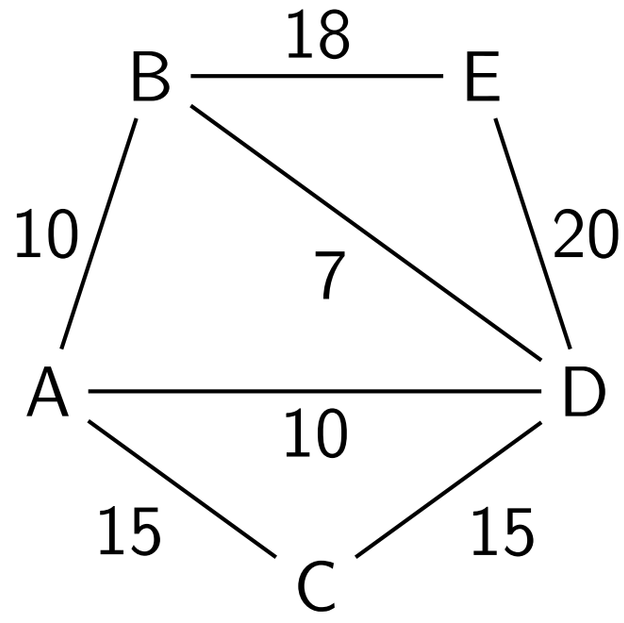
Problème initial

minimiser le coût

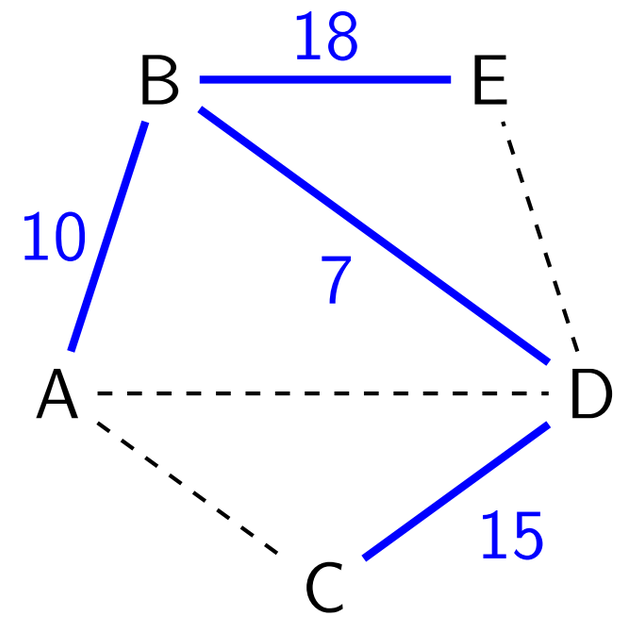
	A	B	C	D	E
A		10	15	10	
B	10			7	18
C	15			15	
D	10	7	15		20
E		18		20	

Problème formel

trouver un arbre couvrant minimal



algorithme : Kruskal



poser des lignes entre :

- A et B
- B et D
- B et E
- C et D

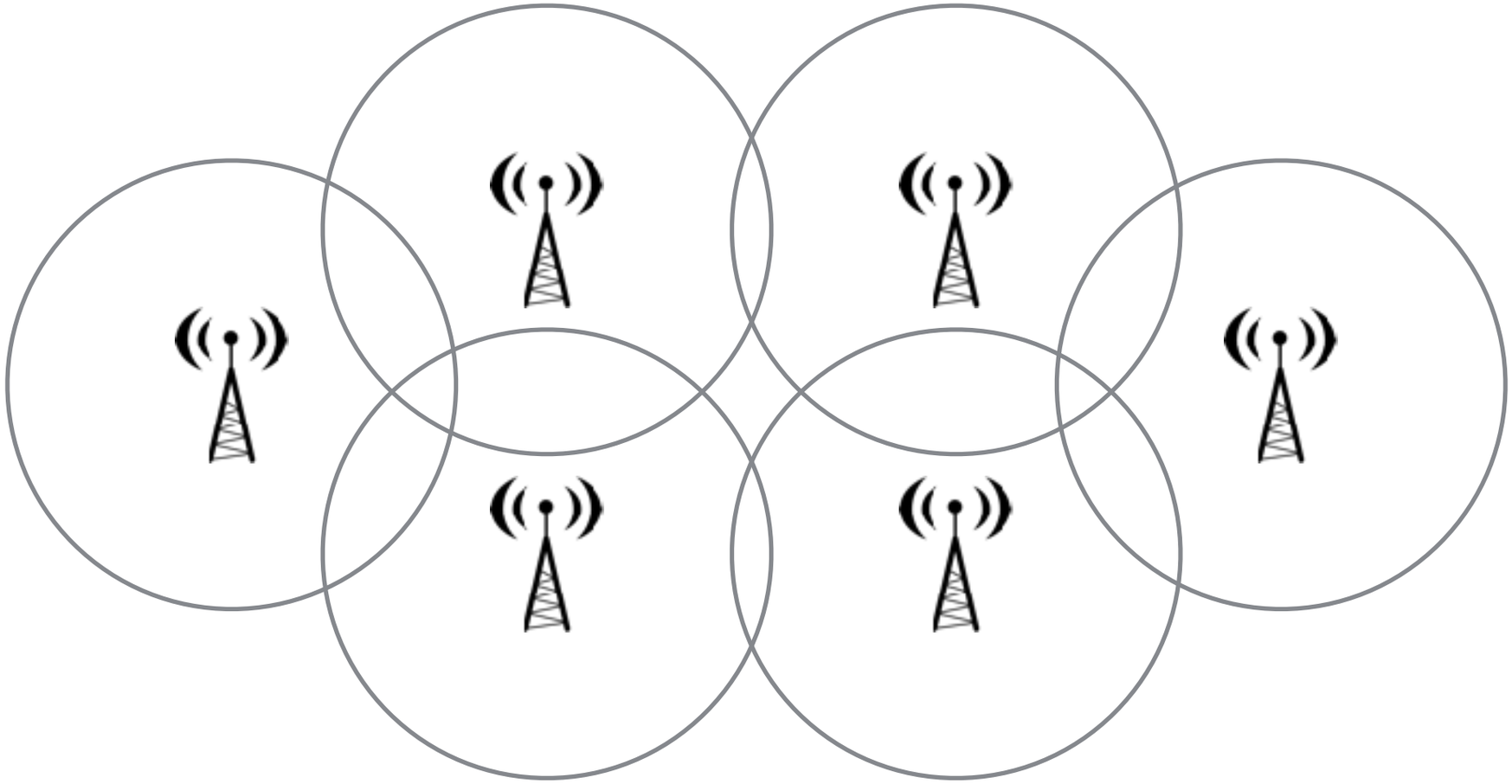
coût total : 10+7+18+15 = 50

Modélisation et résolution d'un problème

- Modélisation
  - Expliciter les suppositions
  - Traduire un **problème** A (le problème initial, sous forme verbale) en un **problème** B (le problème formel, le modèle)
- Résolution
  - Résoudre le problème formel B avec un algorithme / logiciel
- Récupération d'une solution
  - Interpréter une solution de B comme une solution de A

Exemple : affectation de fréquence

Un nombre d'émetteurs est localisé comme dans la figure suivante :

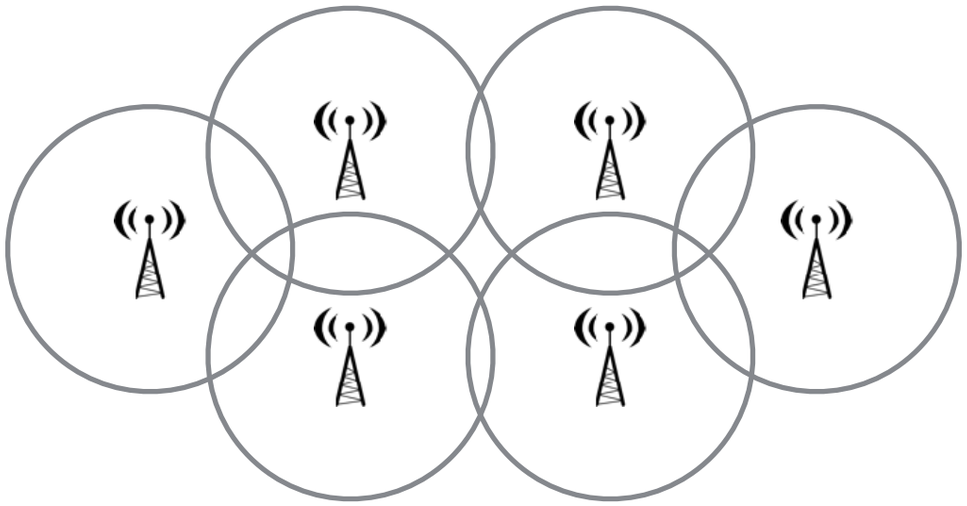


Si deux émetteurs sont trop proches, ils ne peuvent pas transmettre sur la même fréquence à cause d'interférences.

Minimiser le nombre de fréquences utilisées.

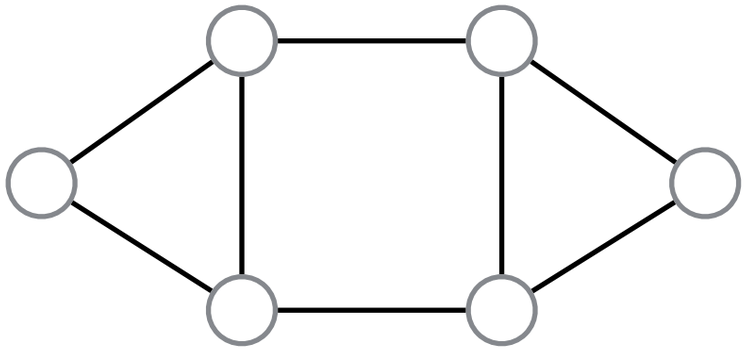
Problème initial

minimiser le nombre de fréquences



Problème formel

trouver une coloration de graphe
avec un nombre minimal de couleurs

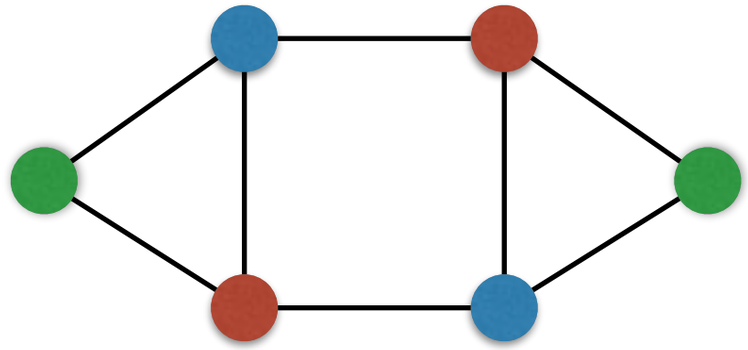


sommet = émetteur

arête = interférence

logiciel dédié

— "solveur"



couleur = fréquence

Solution

affectation :

fréquence 1 (X Mhz)

fréquence 2 (Y Mhz)

fréquence 3 (Z Mhz)

nombre minimal : 3

Modélisation

1. Décider d'un vocabulaire de modélisation
   - Graphes, géométrie, logique, …
2. Chercher les variables de décision et expliciter leur signification
   - Potentiellement plusieurs choix possibles
   - Le ``bon'' choix facilite la prochaine étape
3. Introduire des contraintes qui modélisent le problème
   - Question à se poser : comment les solutions au problème formel B correspondent-elles aux solutions au problème initial A ?

Exemple : programmation linéaire

Une petite entreprise fabrique des robes et des pantalons. Elle dispose de deux machines : une machine pour couper le tissu et une machine pour la couture. Pour fabriquer une robe, il faut une demi heure pour couper le tissu et 20 minutes pour la couture. Pour fabriquer un pantalon, il faut 15 minutes de coupure et une demi heure pour la couture. Le profit d'une robe est de 40 euros et pour un pantalon 50 euros. L'entreprise fonctionne 8 heures par jour.

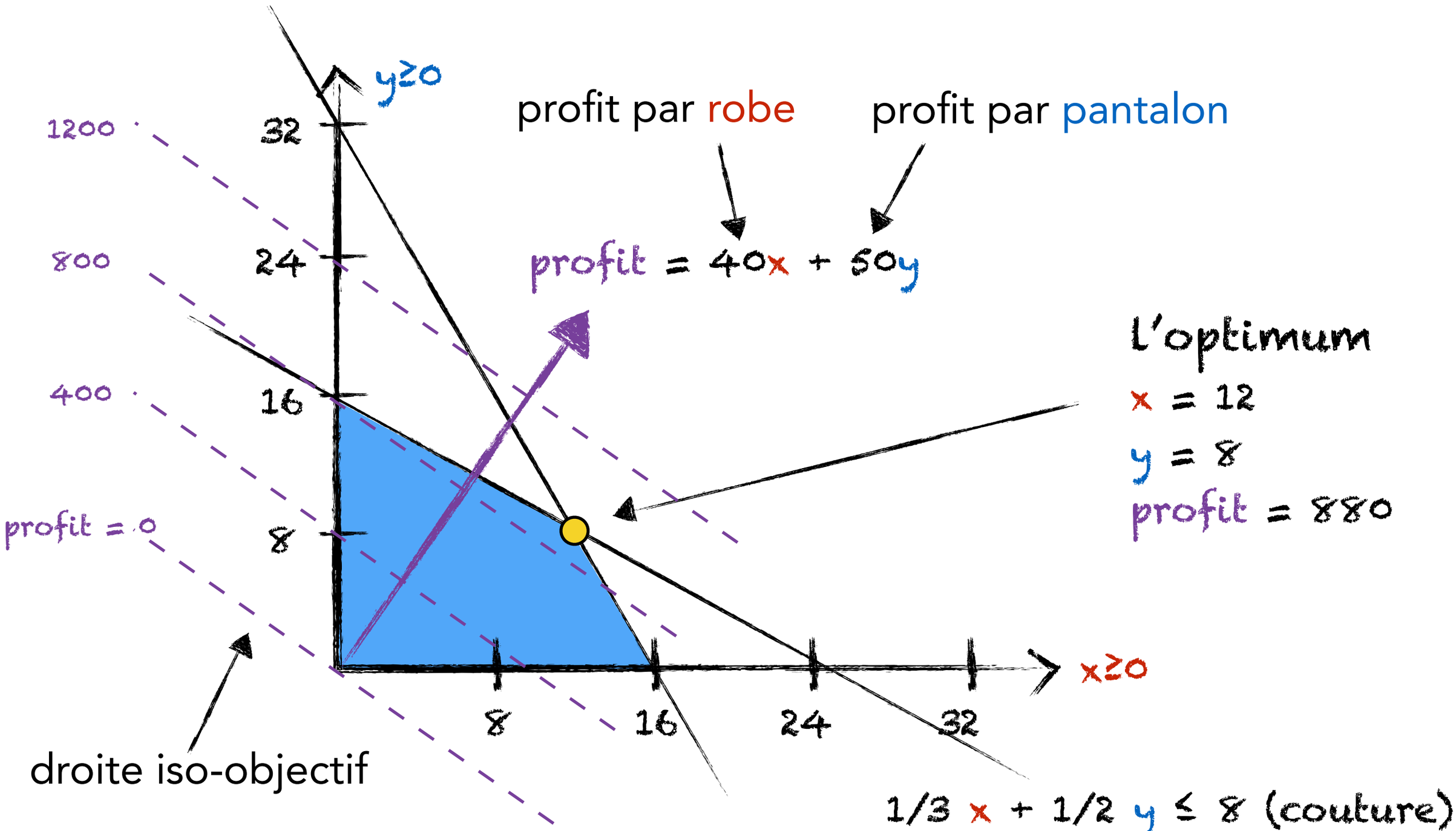
Supposition : on peut coudre jusqu'à 8 heures par jour sans attendre la coupure.

Maximiser le profit par jour.

Robes et pantalons — modélisation

1. Vocabulaire de modélisation
   - Variables réelles et inégalités linéaires
2. Chercher **les variables de décision** et expliciter leur signification
   - Soit x le nombre de robes et y le nombre de pantalons par jour
3. Exprimer les **contraintes** en inégalités linéaires
   - x et y sont non-négatifs : $x, y \geq 0$
   - On coupe au plus 8 heures par jour : $1/2\ x + 1/4\ y \leq 8$
   - On coud au plus 8 heures par jour : $1/3\ x + 1/2\ y \leq 8$

$1/2\ x + 1/4\ y \leq 8$ (coupure)



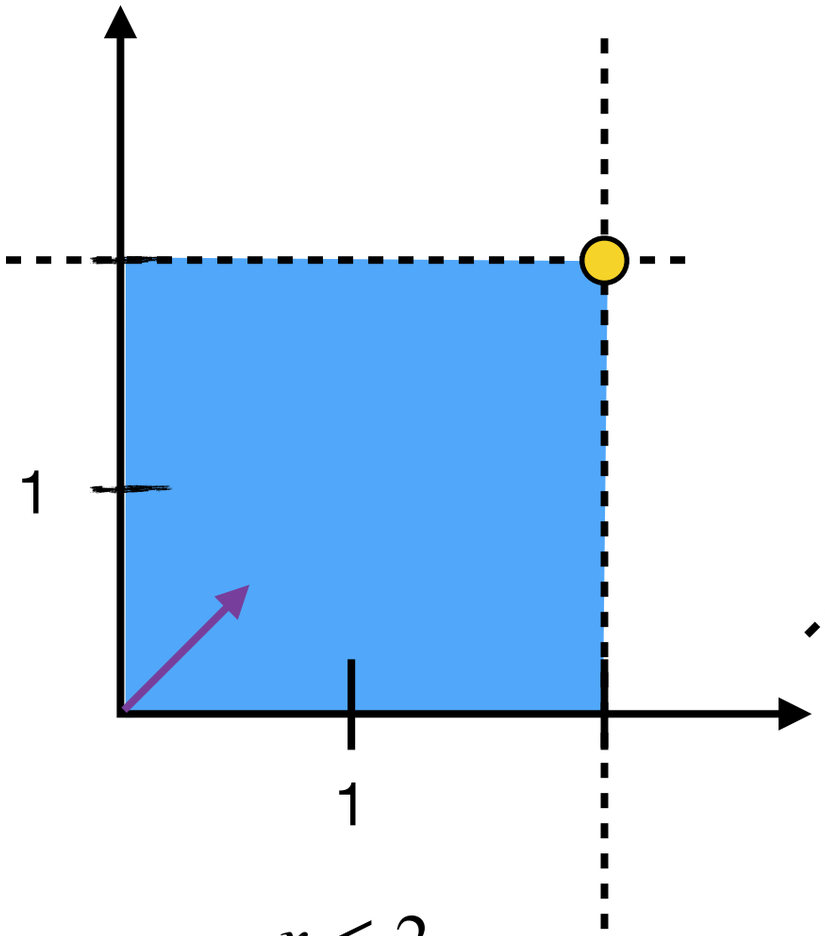
x — nombre de robes par jour

y — nombre de pantalons par jour

solutions admissibles

Sorties possibles

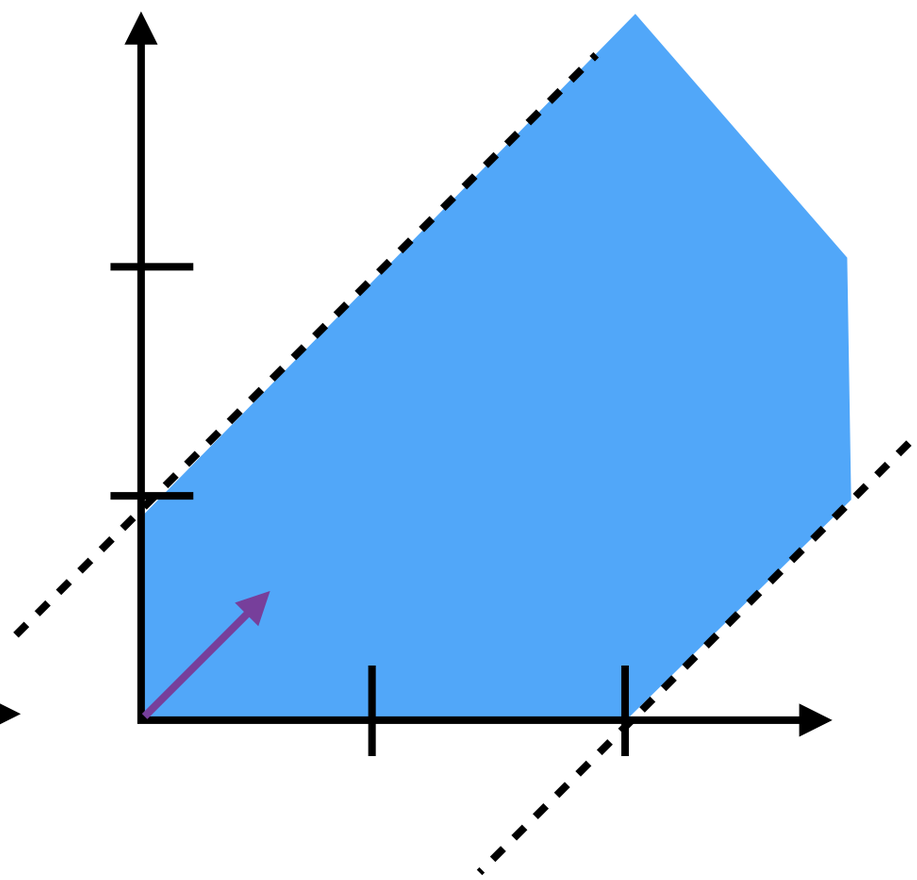
objectif $\max\ x + y$



$x \leq 2$
$y \leq 2$

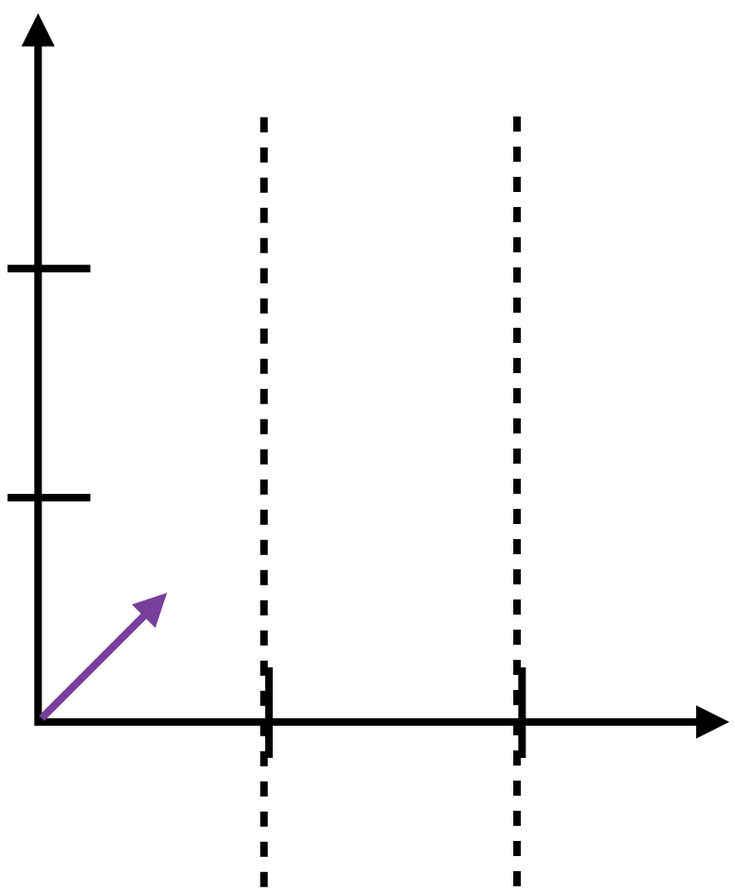
solution admissible optimale

$x = 2, y = 2$



$x - y \geq -1$
$x - y \leq 2$

optimum non borné



$x \leq 1$
$x \geq 2$

pas de solution

La programmation linéaire

- Entrées
  - variables réelles : $x_1, x_2, x_3, \ldots$
  - fonction objectif (linéaire) : $\max \; x_1 + x_2 + 5x_3$
  - équations et inégalités (linéaires) : $x_1 + 7x_2 - 5x_3 \leq 3$
    (non strictes)
- Sorties possibles
  - une solution (admissible) optimale : $x_1 = 2, x_2 = 0, \ldots$
  - ``optimum non borné'' — s'il n'y a pas de valeur optimale finie
  - ``pas de solution (admissible)'' — s'il n'en existe aucune

Résolution efficace en théorie et en pratique

- ***L'algorithme du simplexe*** (Dantzig 1947)
  - efficace et utilisé en pratique
  - exponentiel dans le pire des cas

- Algorithmes de complexité polynomiale
  - méthode de l'ellipsoïde (polynomiale, efficace en théorie)
  - méthodes de points intérieurs
    (polynomiales, efficaces en théorie et en pratique)

- Nous utilisons le solveur **lp_solve** (simplexe)

```
$ sudo apt-get install lp-solve
```

Un peu d'histoire

- Kantorovitch (~1939, URSS), Dantzig (1947, États-Unis)
- "Programmation" = planification
  - le terme date des années 1950 (e.g. programmation musicale)
- Applications importantes
  - planification, logistique, contrôle de la production dans de nombreux domaines : télécom, transport aérien, …
- Kantorovitch lauréat du *"Prix de la Banque de Suède en sciences économiques en mémoire d'Alfred Nobel"* (1975)
  - domaine : théorie de l'allocation optimale des ressources

Exemple : satisfaisabilité booléenne

Vous vous retrouvez devant trois portes. Derrière une des portes, il y a un trésor et derrière les autres ce sont des chimères affamées. Sur chaque porte, il y a un indice. Seul un des indices est vrai et les autres sont faux.

Les indices sont :

porte A : "*le trésor n'est pas ici*"
porte B : "*le trésor n'est pas ici*"
porte C : "*le trésor est derrière la porte B*"

Derrière quelle porte se trouve le trésor ?

Trésor — modélisation

1. Vocabulaire de modélisation
   - variables booléennes et clauses CNF
2. Chercher **les variables de décision** et expliciter leur signification
   - la variable $x_A = 1$ ssi le trésor est derrière la porte A
   - la variable $x_B = 1$ ssi le trésor est derrière la porte B
   - la variable $x_C = 1$ ssi le trésor est derrière la porte C

Trésor — modélisation

3. Exprimer **les contraintes** en logique propositionnelle

Vous vous retrouvez devant trois portes. Derrière une des portes, il y a un trésor et derrière les autres ce sont des chimères affamées. Sur chaque porte, il y a un indice. Seul un des indices est vrai et les autres sont faux.

Les indices sont :

porte A : “*le trésor n'est pas ici*”

porte B : “*le trésor n'est pas ici*”

porte C : “*le trésor est derrière la porte B*”

Derrière quelle porte se trouve le trésor ?

Trésor — modélisation

3. Exprimer **les contraintes** en logique propositionnelle

Vous vous retrouvez devant trois portes. Derrière une des portes, il y a un trésor et derrière les autres ce sont des chimères affamées. Sur chaque porte, il y a un indice. Seul un des indices est vrai et les autres sont faux.

Les indices sont :

porte A : "*le trésor n'est pas ici*"
porte B : "*le trésor n'est pas ici*"
porte C : "*le trésor est derrière la porte B*"

Derrière quelle porte se trouve le trésor ?

$(x_A \vee x_B \vee x_C) \wedge (\neg x_A \vee \neg x_B) \wedge (\neg x_A \vee \neg x_C) \wedge (\neg x_B \vee \neg x_C)$

Trésor — modélisation

3. Exprimer **les contraintes** en logique propositionnelle

Vous vous retrouvez devant trois portes. Derrière une des portes, il y a un trésor et derrière les autres ce sont des chimères affamées. Sur chaque porte, il y a un indice. Seul un des indices est vrai et les autres sont faux.

Les indices sont :

porte A : "*le trésor n'est pas ici*" $\neg X_A$

porte B : "*le trésor n'est pas ici*" $\neg X_B$

porte C : "*le trésor est derrière la porte B*" X_B

Derrière quelle porte se trouve le trésor ?

$$(X_A \vee X_B \vee X_C) \wedge (\neg X_A \vee \neg X_B) \wedge (\neg X_A \vee \neg X_C) \wedge (\neg X_B \vee \neg X_C)$$

Trésor — modélisation

3. Exprimer **les contraintes** en logique propositionnelle

Vous vous retrouvez devant trois portes. Derrière une des portes, il y a un trésor et derrière les autres ce sont des chimères affamées. Sur chaque porte, il y a un indice. Seul un des indices est vrai et les autres sont faux.

Les indices sont :

porte A : "*le trésor n'est pas ici*" $\neg X_A$

porte B : "*le trésor n'est pas ici*" $\neg X_B$

porte C : "*le trésor est derrière la porte B*" X_B

Derrière quelle porte se trouve le trésor ?

$(X_A \vee X_B \vee X_C) \wedge (\neg X_A \vee \neg X_B) \wedge (\neg X_A \vee \neg X_C) \wedge (\neg X_B \vee \neg X_C)$

$(\neg X_A \vee \neg X_B \vee X_B) \wedge (X_A \vee X_B) \wedge (X_A \vee \neg X_B) \wedge (X_B \vee \neg X_B)$

Trésor — modélisation

$$(X_A \vee X_B \vee X_C) \wedge (\neg X_A \vee \neg X_B) \wedge (\neg X_A \vee \neg X_C) \wedge (\neg X_B \vee \neg X_C) \wedge$$
$$(\neg X_A \vee \neg X_B \vee X_B) \wedge (X_A \vee X_B) \wedge (X_A \vee \neg X_B) \wedge (X_B \vee \neg X_B)$$

Solveur

$X_A = 1$

$X_B = 0$

$X_C = 0$

Satisfaisabilité booléenne

- Entrées
  - variables booléennes : $x_1, x_2, x_3, \ldots$
  - clauses (disjonctions) : $(x_1 \vee x_2), (x_1 \vee \neg x_2 \vee \neg x_3), \ldots$
- Sortie
  - une affectation satisfaisante : $x_1 = 1, x_2 = 0, \ldots$
    (qui rend toutes les clauses vraies)
  - ``non satisfaisable'' — s'il n'en existe aucune

Résolution quasi-efficace en pratique

- Problème NP-difficile !
  - On ne connait pas d'algorithme polynomial pour SAT
  - Il n'en existe aucun, si P ≠ NP !
- Mais il y a des solveurs qui fonctionnent (souvent) bien en pratique :

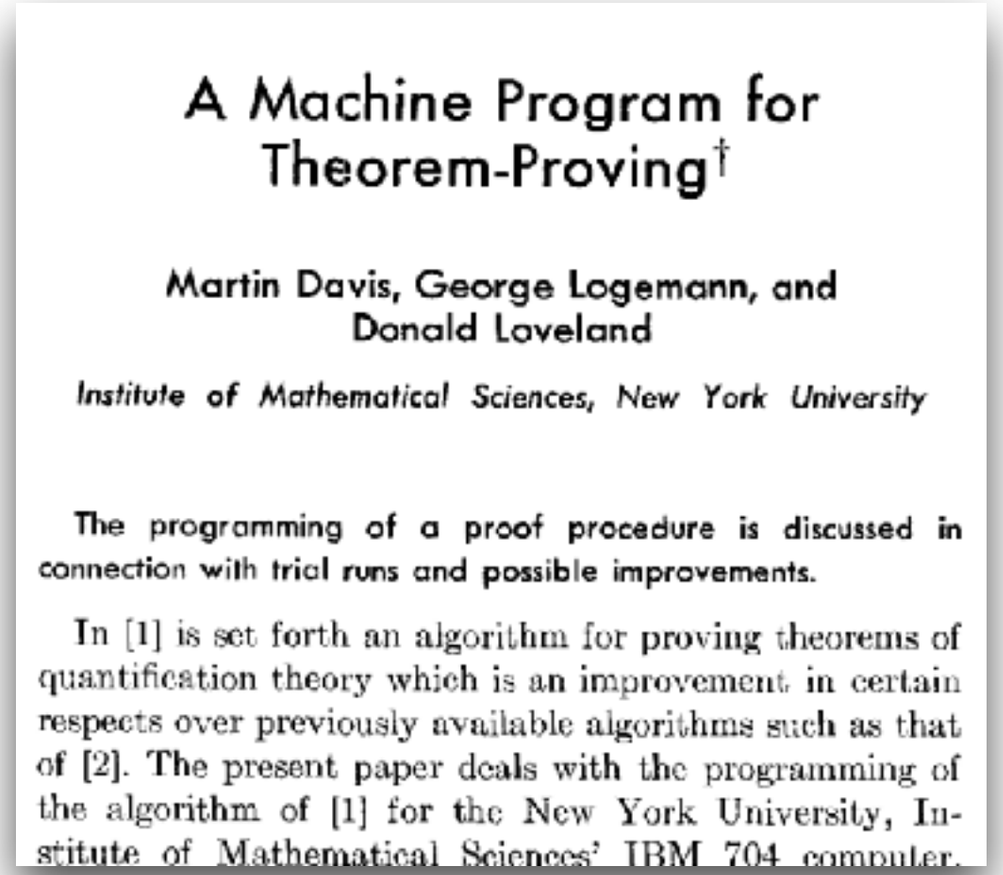
années	variables	clauses
1960-70	10	100
1980-95	100	1 000
1995-00	1 000	100 000
2000-10	100 000	1 000 000
2010-	> 1 000 000	> 5 000 000

- Nous utilisons le solveur **minisat**

```
$ sudo apt-get install minisat
```

Algorithme : DPLL

- **D**avis & **P**utnam (1960)
- Davis, **L**ogeman, **L**oveland (1962)
- Applications importantes
  - vérification formelle des circuits
  - planification (emploi du temps)
  - ordonnancement



A Machine Program for Theorem-Proving†

Martin Davis, George Logemann, and Donald Loveland

Institute of Mathematical Sciences, New York University

The programming of a proof procedure is discussed in connection with trial runs and possible improvements.

In [1] is set forth an algorithm for proving theorems of quantification theory which is an improvement in certain respects over previously available algorithms such as that of [2]. The present paper deals with the programming of the algorithm of [1] for the New York University, Institute of Mathematical Sciences' IBM 704 computer.

- Application amusante

  [La plus grosse preuve de l'histoire des mathématiques]

  - construction et vérification d'une preuve de 200 To en utilisant 1 solveur SAT, 800 coeurs et 2 jours…

Optimisation

- Recherche d'un élément optimal dans un espace de configurations
  - C — l'espace de configurations (arbres couvrants, solutions admissibles)
  - $f\colon C \to \mathbb{R}$ — la fonction objectif
  - type d'optimisation — $\max_{x \in C} f(x)$ ou $\min_{x \in C} f(x)$
- Résolution exacte
  - en temps polynomial (arbre couvrant minimal, programmation linéaire)
  - en temps exponentiel (coloration de graphe, SAT)
- Résolution approchée
- Heuristiques (algo glouton, recherche locale)

Contenu du cours

- Méthodes
  - L'algorithme du simplexe (programmation linéaire)
  - L'algorithme DPLL (SAT) et CDCL (conflict driven clause learning)
  - Solveurs — logiciels optimisés pour la résolution de problèmes
- Modélisation
  - Traduire un problème sous forme verbale en un programme linéaire (LP) ou en une formule CNF (SAT)
- Optimisation
  - Optimiser une fonction linéaire sous des contraintes linéaires