Algorithmes d’approximation

de la théorie à la pratique

Nicolas Schabanel

CNRS - LIP - École normale supérieure de Lyon

Au début de l’informatique (avant 1971-1972)

Problèmes d’optimisation

Couplage maximum

Voyageur de commerce

Arbre couvrant de poids minimum

SAT / Max-SAT

Couverture par ensembles

Flot maximum

Sac-à-dos

Clique maximum

Au début de l'informatique (avant 1971-1972)

Problèmes d'optimisation

Couplage maximum

Voyageur de commerce

Arbre couvrant de poids minimum

SAT / Max-SAT

Couverture par ensembles

Flot maximum

Sac-à-dos

$\in$ **P**

Clique maximum

Cook-Levin (1971)
Karp (1972)

Problèmes d'optimisation

Clique maximum

Couverture par ensembles

NP-difficiles

SAT / Max-SAT

Voyageur de commerce

Sac-à-dos

Arbre couvrant de poids minimum

Flot maximum

$\in$ P

Couplage maximum

Que faire lorsque le problème est NP-difficile ?

Première idée: des heuristiques

Ex: Voyageur de commerce

- **glouton :** commencer avec deux villes et ajouter la ville qui étend le moins le tour
- **recherche locale :** commencer avec un tour aléatoire et échanger des villes tant que ça diminue la distance
- **recherche exhaustive :** programmation dynamique, branch & bound,...

Comment évaluer les performances ?

Comparer les heuristiques ?

Sur quelles instances ?

Des instances particulières ?

Des instances aléatoires ?

Ex. d'instances aléatoires : 3SAT

φ = *Clause*$_1$ **et** *Clause*$_2$ **et** ... **et** *Clause*$_m$

Clause$_k$ = x_{i1} **ou** x_{i2} **ou** x_{i3}, avec x_{ij} aléatoire

Ex. d'instances aléatoires : 3SAT

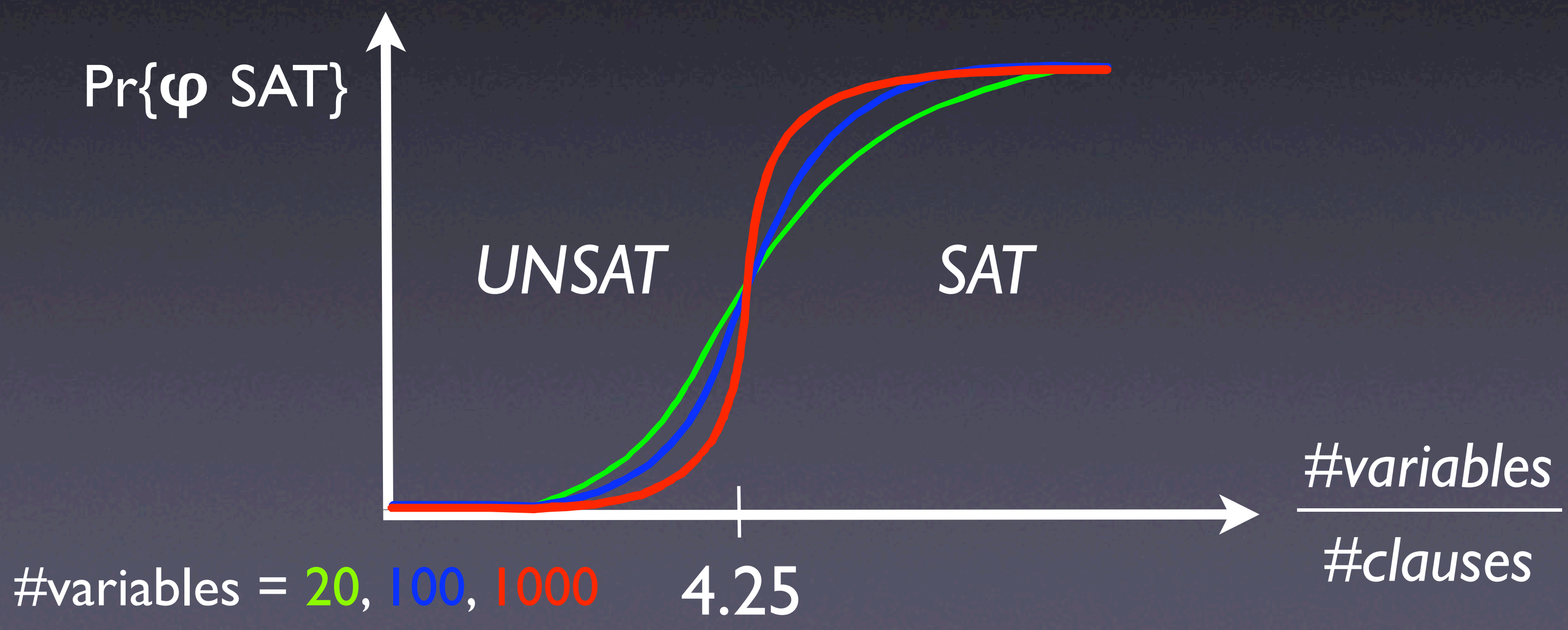
φ = *Clause$_1$* **et** *Clause$_2$* **et** ... **et** *Clause$_m$*

Clause$_k$ = x_{i1} **ou** x_{i2} **ou** x_{i3}, avec x_{ij} aléatoire

Ex. d'instances aléatoires : 3SAT

φ = *Clause*$_1$ **et** *Clause*$_2$ **et** ... **et** *Clause*$_m$

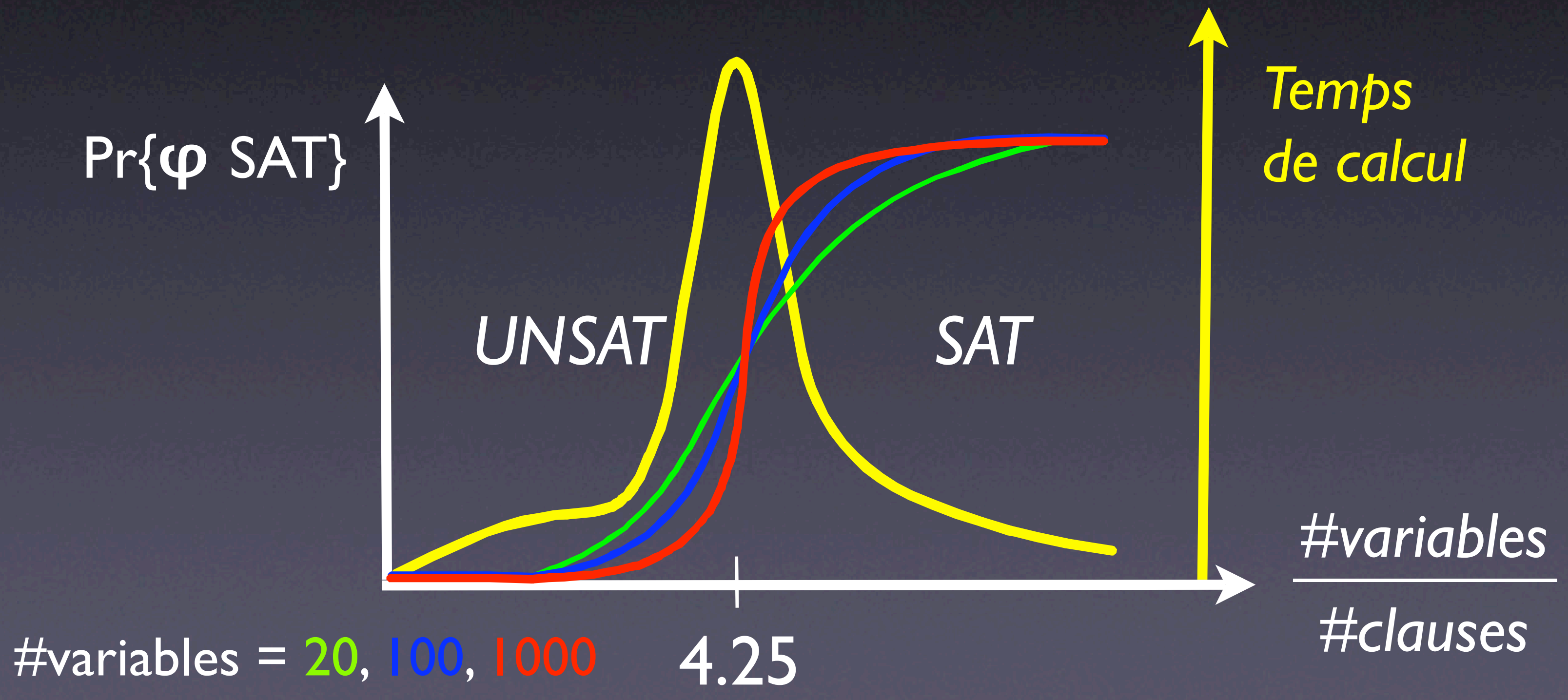
Clause$_k$ = x_{i_1} **ou** x_{i_2} **ou** x_{i_3}, avec x_{i_j} aléatoire



Ex. d'instances aléatoires : 3SAT

φ = Clause_1 **et** Clause_2 **et** ... **et** Clause_m

Clause_k = x_{i1} **ou** x_{i2} **ou** x_{i3}, avec x_{ij} aléatoire



Comparaison sur des instances aléatoires n'est pas si aléatoire que ça !

Début de l'algorithmique (avant 1965)

Différents algorithmes pour le tri...

... comment les comparer ?

1965

Tout algorithme de tri utilise au moins ***n* log *n*** comparaisons.

Début de l'algorithmique (avant 1965)

Différents algorithmes pour le tri...

... comment les comparer ?

1965

Tout algorithme de tri utilise au moins ***n* log *n*** comparaisons.

n* log *n = une estimation *absolue* de l'optimum, *indépendante* des algorithmes et permet de les comparer

Nécessité d'une estimation de **l'optimum**

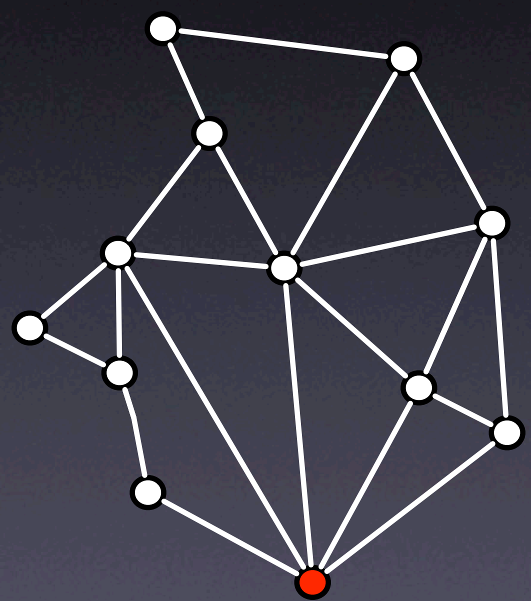
Valables pour toutes les instances

qui permettra de :

- Comparer les heuristiques à cette estimation
- Concevoir de nouvelles heuristiques
- Prouver des garanties de performances

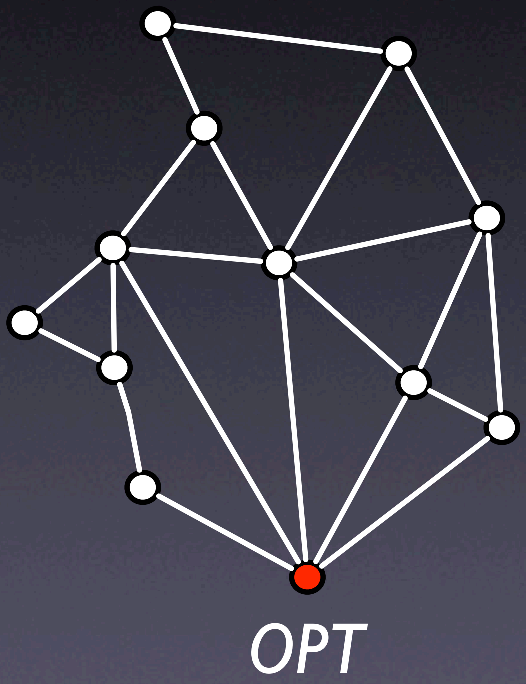
Principe général

$\in$ **P**



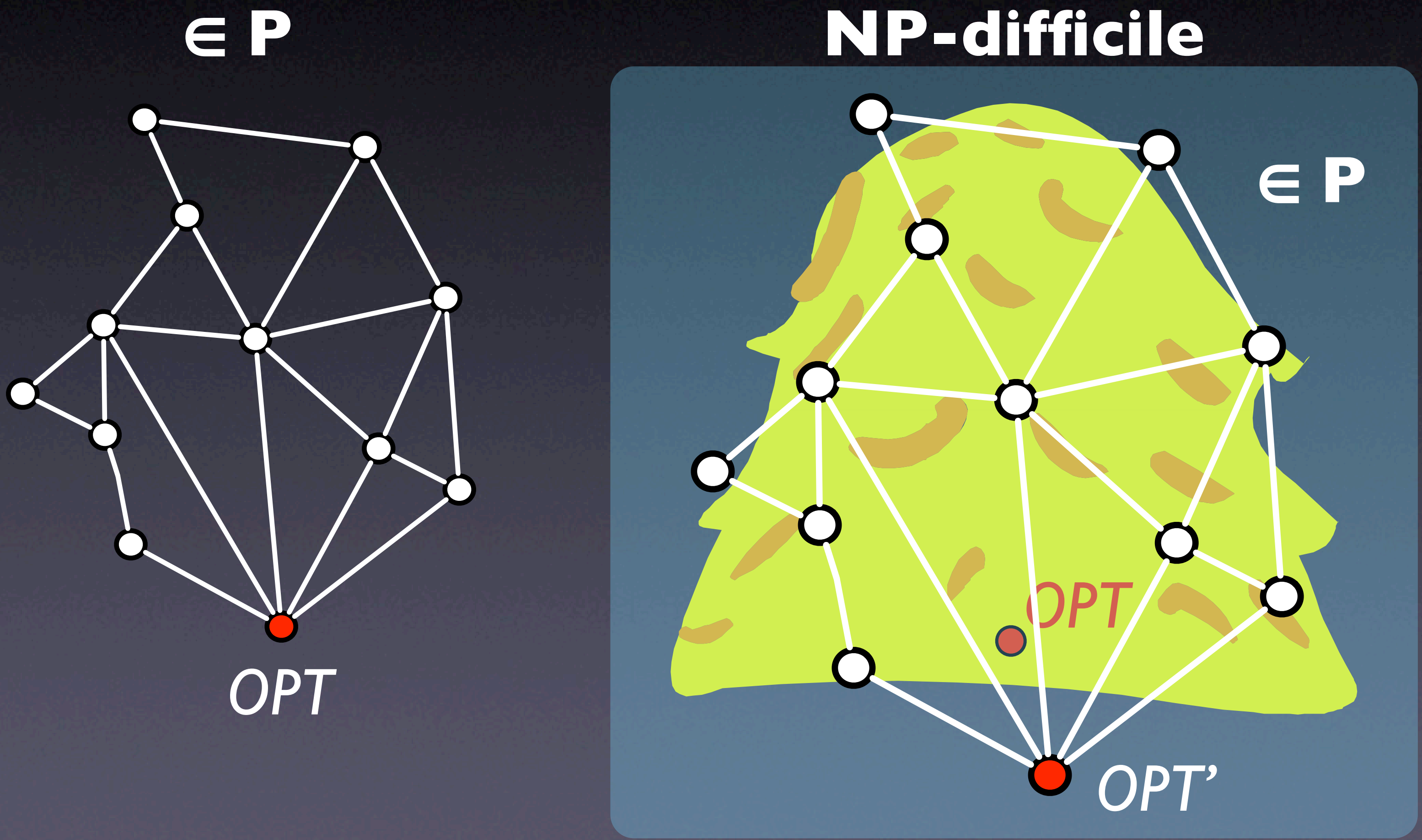
Principe général

∈ P

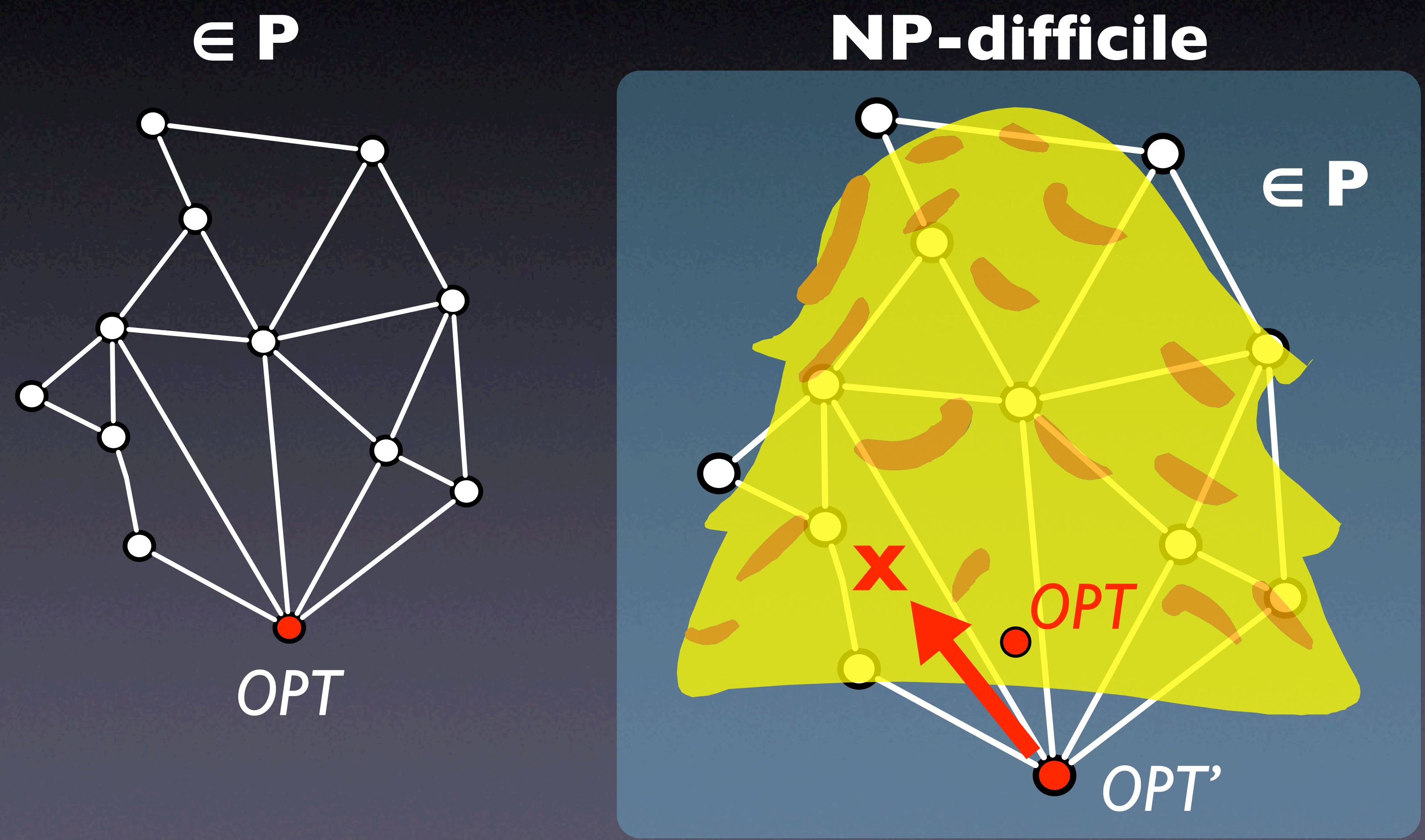


NP-difficile

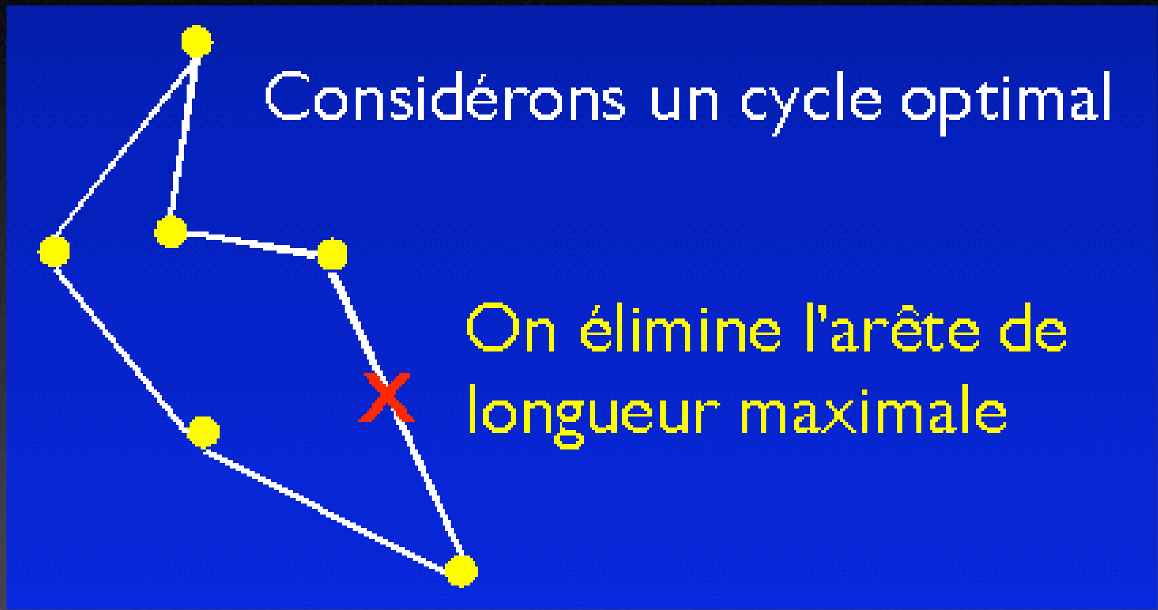
Principe général



Principe général



Voyageur de commerce



On obtient un arbre courant !

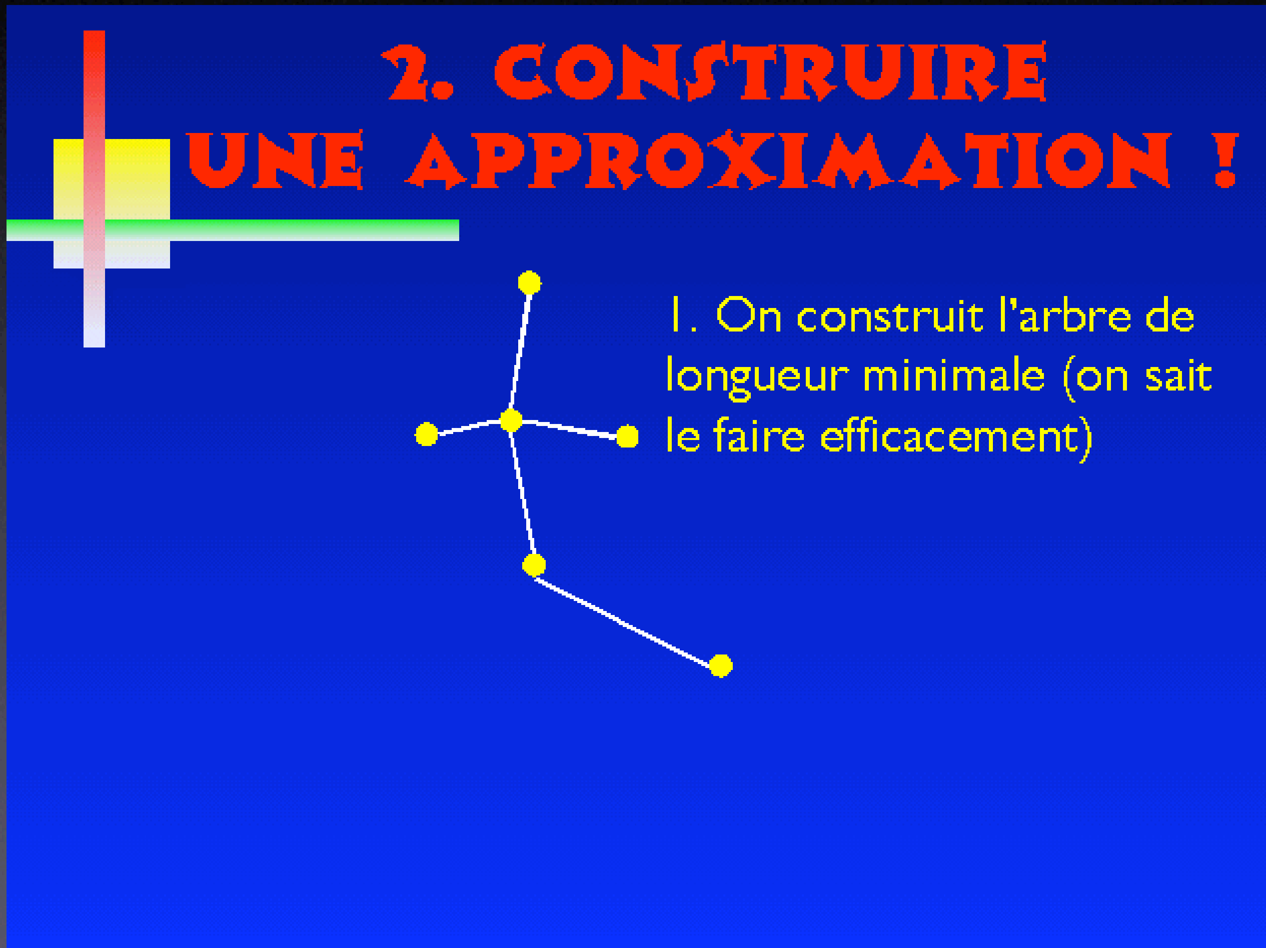
Arbre couvrant de poids minimimum ≤ *OPT*

L'arbre couvrant de poids min est-il un bon minorant ?

À combien est-il de *OPT* ?

Peut-on l'utiliser pour concevoir un nouvel algorithme ?

Une 2-approximation



Une 2-approximation

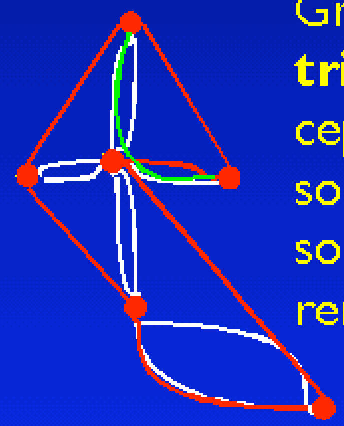
2. CONSTRUIRE UNE APPROXIMATION !

1. On construit l'arbre de longueur minimale (on sait le faire efficacement)
2. On duplique les arêtes

On obtient alors un cycle de **longueur inférieure au double de l'optimal** mais où *les sommets apparaissent plusieurs fois*

Une 2-approximation

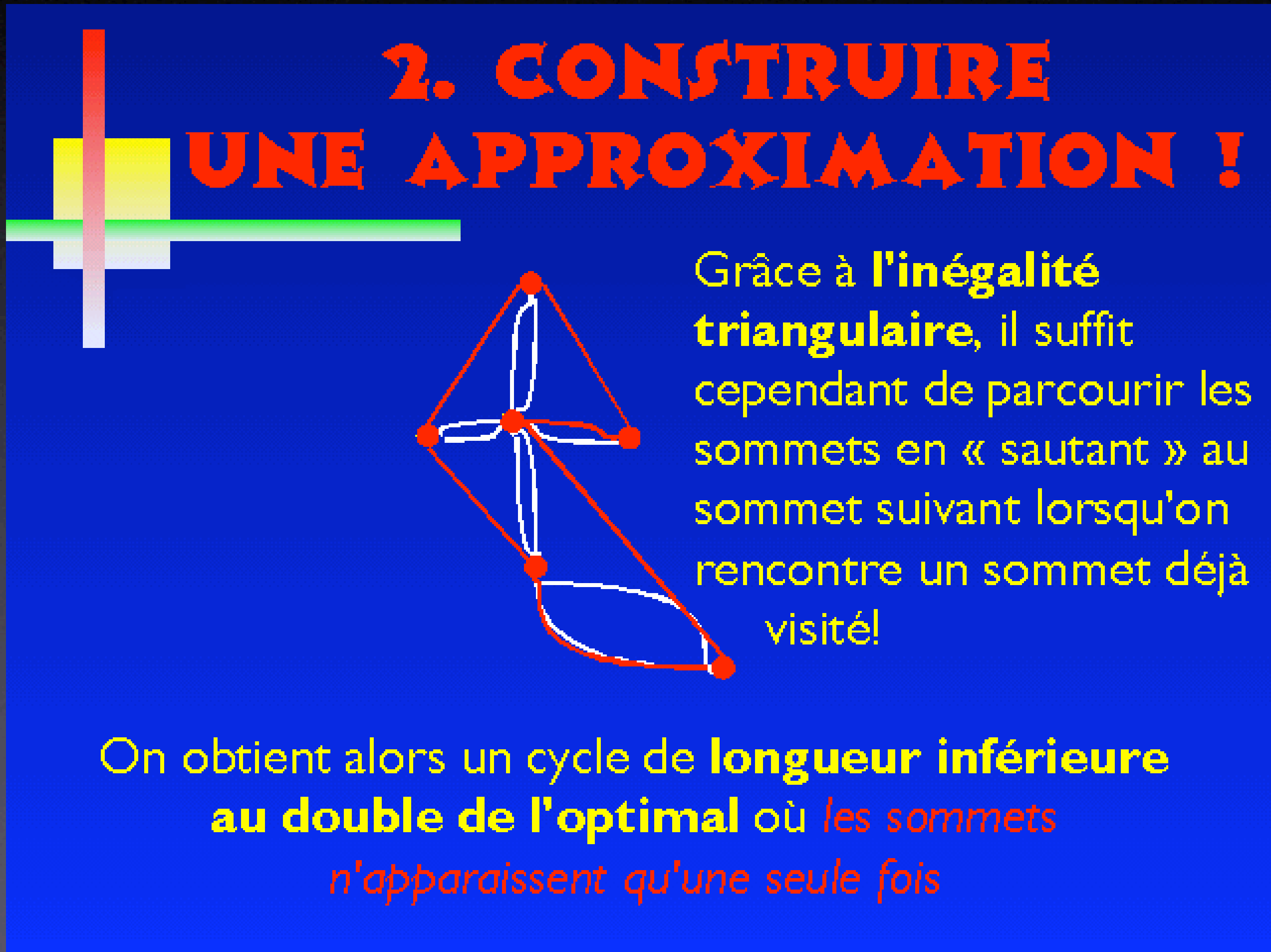
2. CONSTRUIRE UNE APPROXIMATION !



Grâce à **l'inégalité triangulaire**, il suffit cependant de parcourir les sommets en « sautant » au sommet suivant lorsqu'on rencontre un sommet déjà visité!

On obtient alors un cycle de **longueur inférieure au double de l'optimal** mais où *les sommets apparaissent plusieurs fois*

Une 2-approximation



C'est une 2-approximation !

$$\text{Solution} \leq \begin{array}{c}\mathbf{2} \cdot \text{Arbre} \\ \text{couvrant de} \\ \text{poids minimum}\end{array} \leq \mathbf{2} \cdot \mathit{OPT}$$

C'est un bon minorant !

$$\frac{OPT}{2} \leq \text{Arbre couvrant de poids minimum} \leq OPT \leq 2 \times \text{Arbre couvrant de poids minimum}$$

À quoi sert un bon minorant ?

Une évaluation impartiale d'une heuristique

Concevoir de nouveaux algorithmes dont les performances sont prouvées

Développer des stratégies exhaustives : type Branch & Bound

Peut-on faire beaucoup mieux ?

(1999)
Impossible d'approcher à moins de

$2805 / 2804 - \epsilon = 1.0003563 - \epsilon$

pour tout $\epsilon > 0$, même si les distances sont 1 ou 2 seulement, sauf si **P = NP**

Difficulté des problèmes d'optimisation (1972)

Problèmes d'optimisation

Clique maximum

Couverture par ensembles

NP-difficiles

SAT / Max-SAT

Voyageur de commerce

Sac-à-dos

Arbre couvrant de poids minimum

Flot maximum

$\in$ P

Couplage maximum

Difficulté des problèmes d'optimisation (1992-2001)

Inapproximables à o(n^{ϵ})
Clique maximum

Inapproximables à o(log *n*)
Couverture par ensembles

Inapproximables à < Cte
Max-SAT, Voyageur de commerce

Approximables ϵ près pour tout ϵ >0
Sac-à-dos

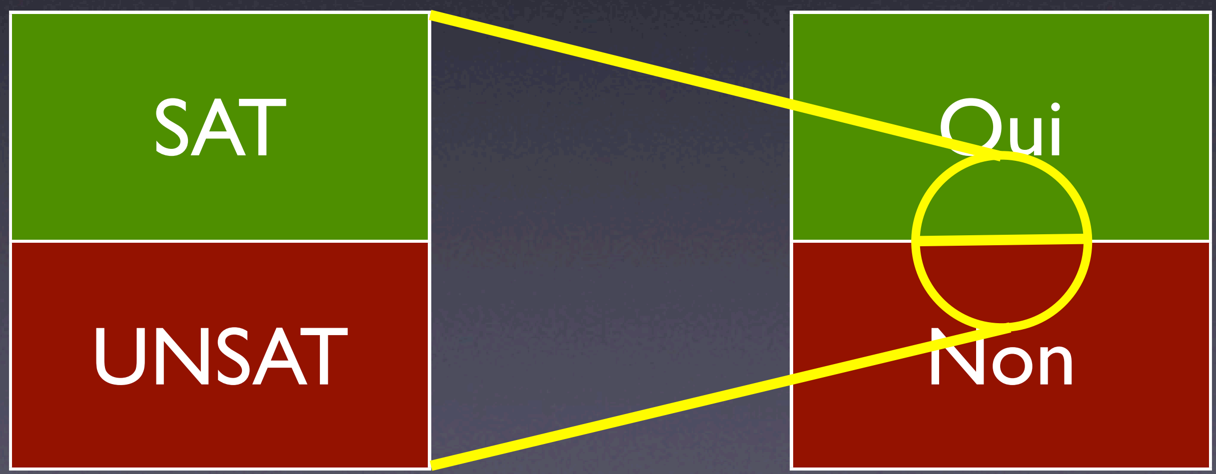
$\in$ P

Prouver la **NP**-difficulté

Réduction depuis SAT

SAT

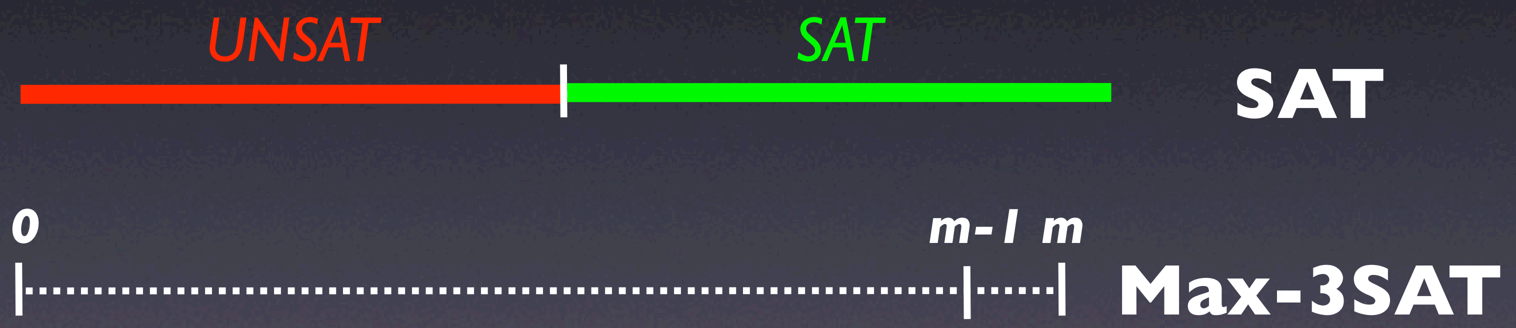
“Existe-t-il une solution de coût ≤ ***k*** ?”



Difficulté de l'approximation

Réduction écartante

Il faut écarter les coûts des solutions optimales



Difficulté de l'approximation

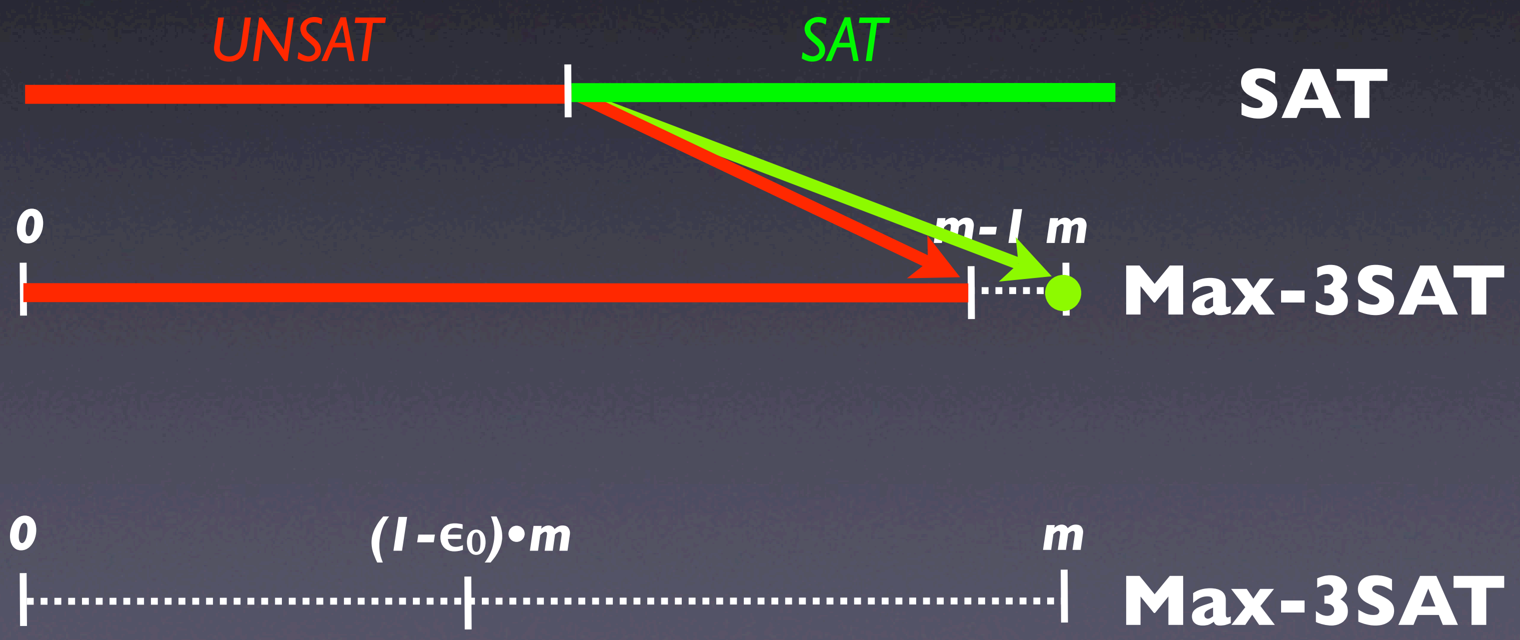
Réduction écartante

Il faut écarter les coûts des solutions optimales

Difficulté de l'approximation

Réduction écartante

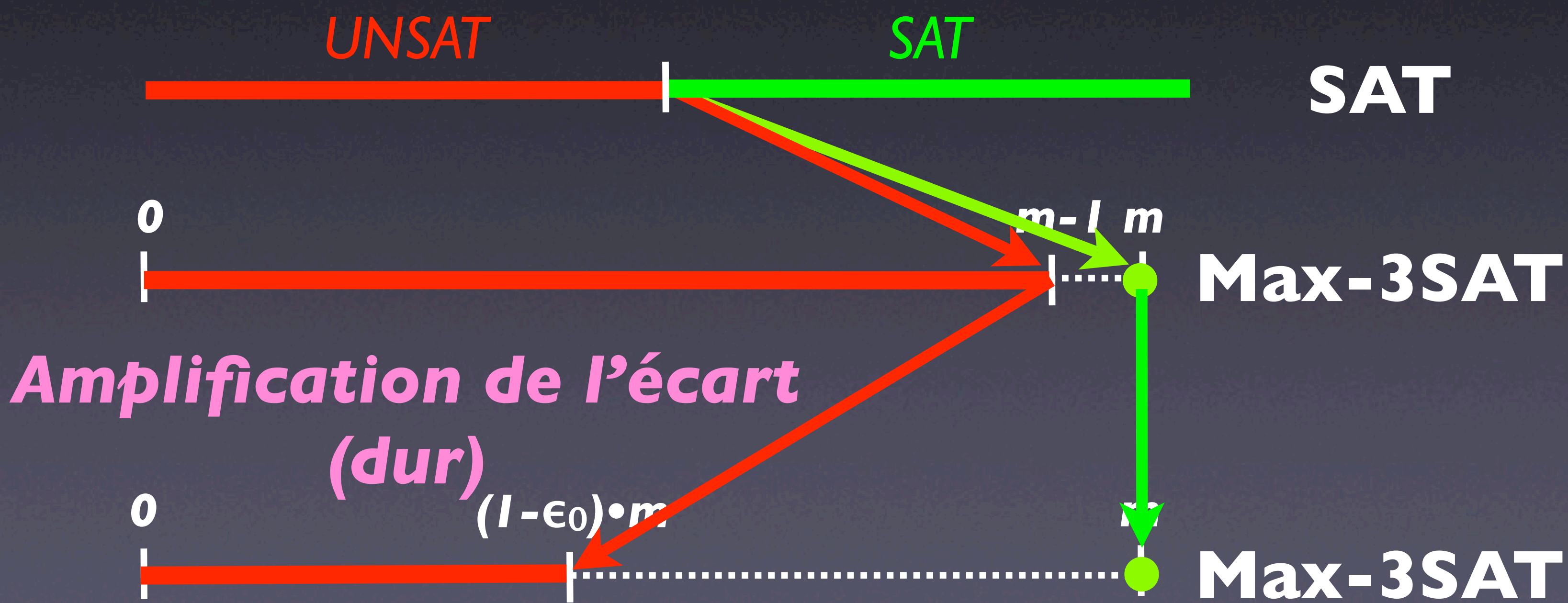
Il faut écarter les coûts des solutions optimales



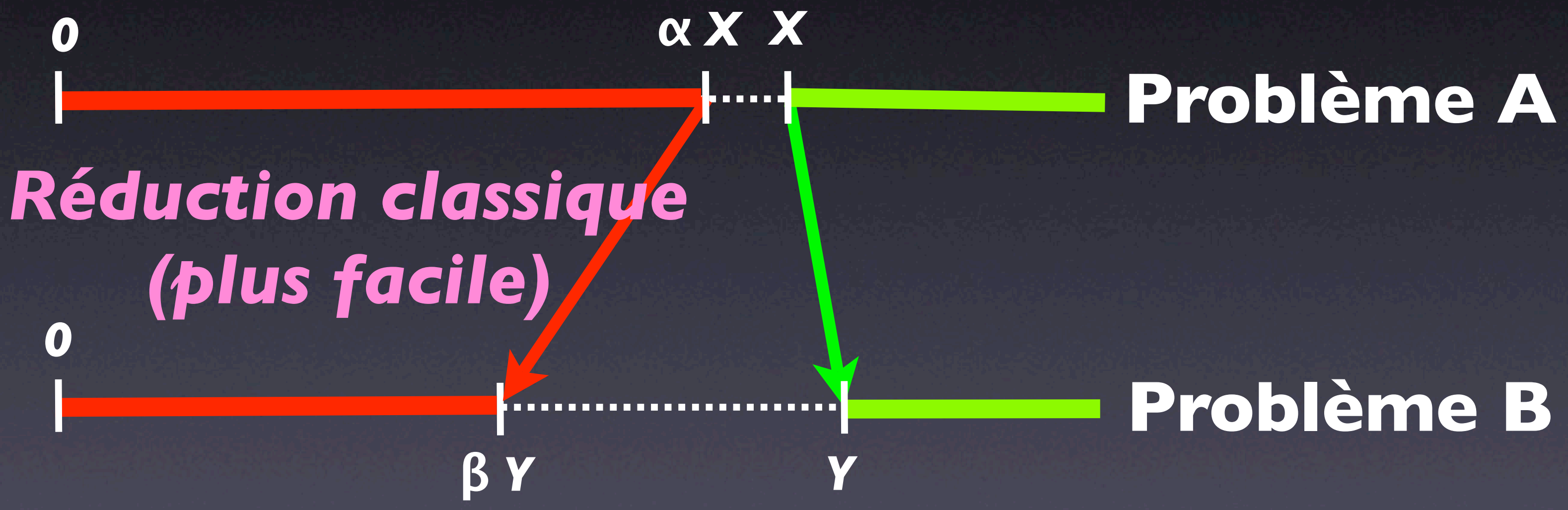
Difficulté de l'approximation

Réduction écartante

Il faut écarter les coûts des solutions optimales



Réductions d'écart



Difficulté des problèmes d'optimisation (1992-2001)

Inapproximables à o(n^ϵ)
Clique maximum

Inapproximables à o(log *n*)
Couverture par ensembles

Inapproximables à < Cte
Max-SAT, Voyageur de commerce

Approximables ϵ près pour tout ϵ >0
Sac-à-dos

$\in$ P

Conclusion

Principe

- Relâcher les contraintes et obtenir un minorant du coût minimum
- En déduire un algorithme à prouver

Inapproximabilité

- Réduction polynomial créant/conservant un **“gap”**

Et en pratique ?

Le minorant du coût optimum

- Estimation *fidèle* du coût optimum
- Idées pour construire des heuristiques
- Un minorant peut-être meilleur que l'algorithme pourrait le faire croire

Ex : *le problème du vecteur le plus court d'un réseau*

$$\frac{OPT}{\boldsymbol{n}} \leq \quad Minorant \quad \leq \quad OPT \quad \leq \mathbf{2^n} \cdot Minorant$$

Merci !