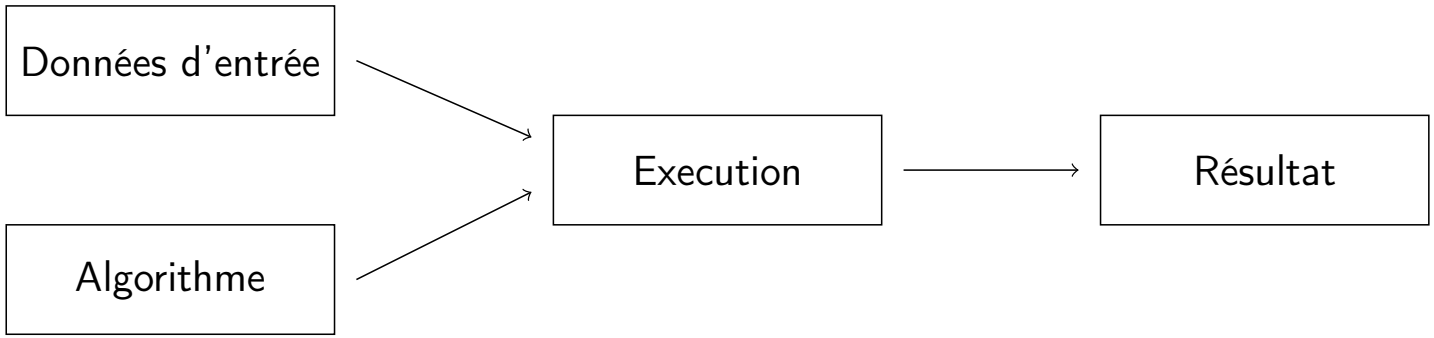
Algorithmes "Online" : introduction à la prise de décisions dans un monde incertain

Thomas Lambert (thomas.lambert 'at' inria.fr)

Table des matières

Modèle offline

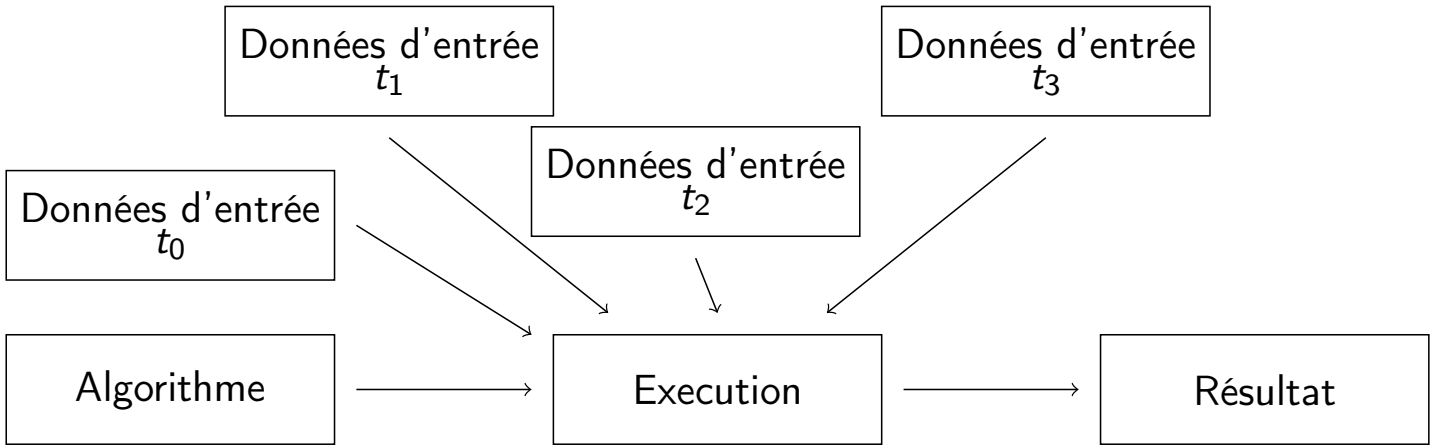
- **Modèle classique (offline) :**



- **Algorithmes offlines :**
  - Toutes les entrées sont connues
  - On est parfois obligé de faire des prédictions.

Modèle online

- **Modèle online** :



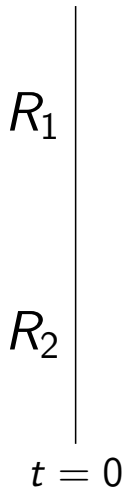
- **Algorithmes onlines** :
  - Décision prises durant l'exécution.

Une application : l'ordonnancement

- **L'ordonnancement** c'est la distribution de **tâches** sur des **ressources** en essayant d'atteindre des **objectifs**.
  - Tâches : programme à exécuter, cours, bus de transport, requêtes sur des serveurs, étapes d'un projet de construction...
  - Ressources : ordinateurs, travailleurs, routes, salles...
  - Objectifs : Minimiser le temps de calcul, la consommation d'énergie, le retard, l'inaction...
- Un des problèmes lorsqu'on fait de l'ordonnancement c'est qu'on ne connaît pas toutes les tâches au début de l'exécution (exemple : requêtes sur un serveur).
- Quelles conséquences ?

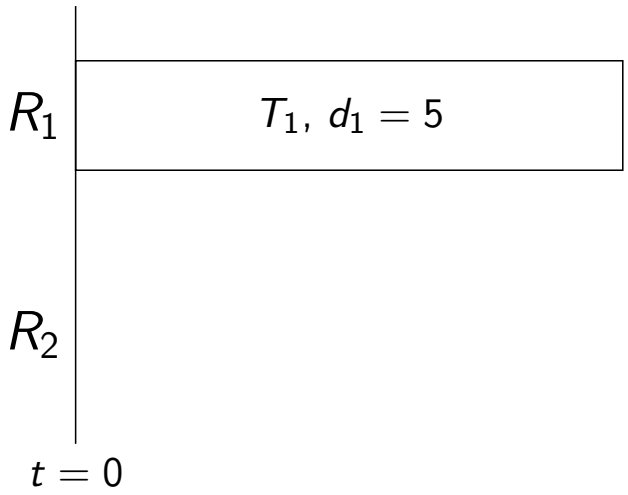
Le coût de l'ignorance

- Deux ressources, des tâches de durées variables qui arrivent à des dates inconnues.
- Algorithme : on lance la tâche arrivée le plus tôt sur la première machine disponible.



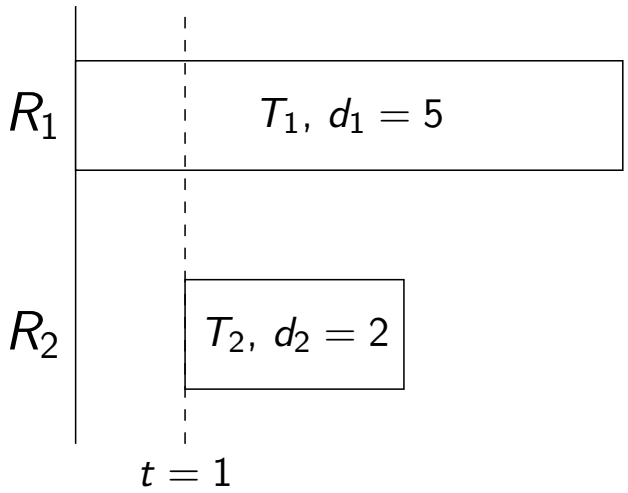
Le coût de l'ignorance

- Deux ressources, des tâches de durées variables qui arrivent à des dates inconnues.
- Algorithme : on lance la tâche arrivée le plus tôt sur la première machine disponible.



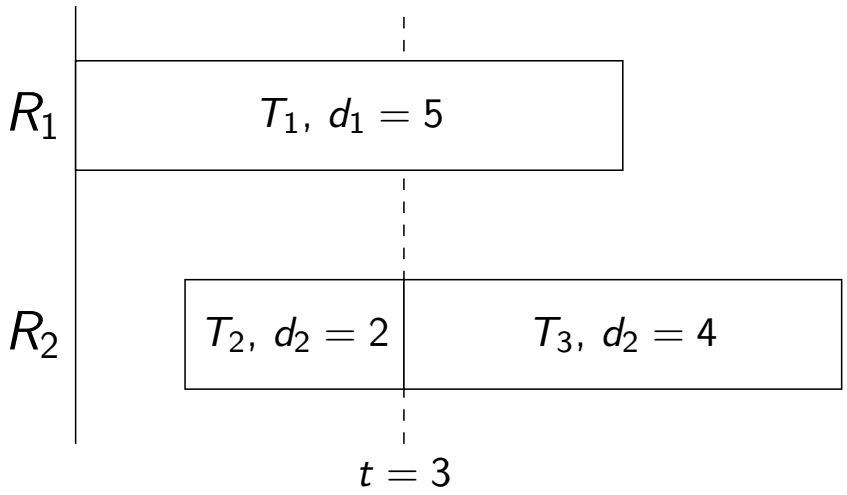
Le coût de l'ignorance

- Deux ressources, des tâches de durées variables qui arrivent à des dates inconnues.
- Algorithme : on lance la tâche arrivée le plus tôt sur la première machine disponible.



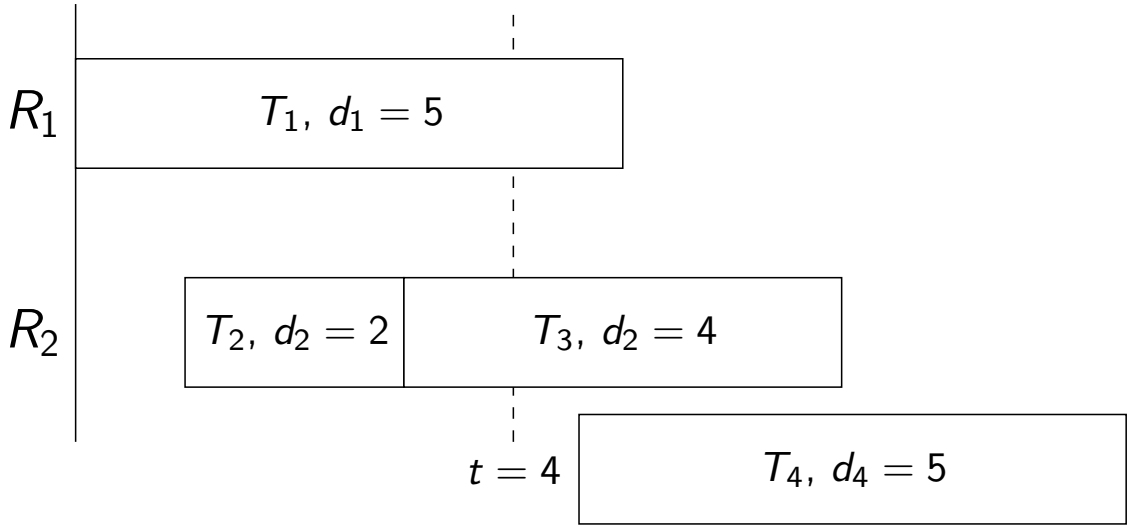
Le coût de l'ignorance

- Deux ressources, des tâches de durées variables qui arrivent à des dates inconnues.
- Algorithme : on lance la tâche arrivée le plus tôt sur la première machine disponible.



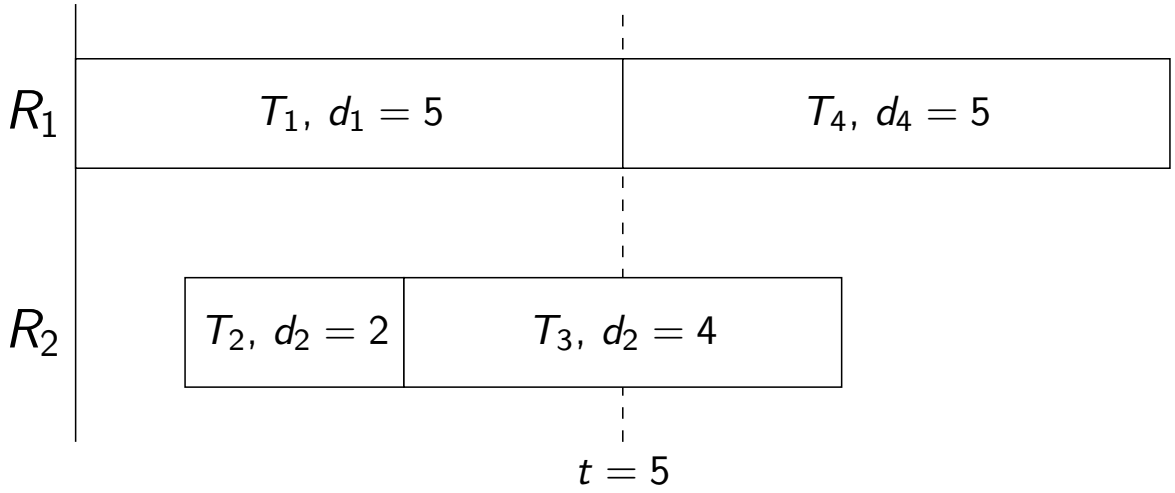
Le coût de l'ignorance

- Deux ressources, des tâches de durées variables qui arrivent à des dates inconnues.
- Algorithme : on lance la tâche arrivée le plus tôt sur la première machine disponible.



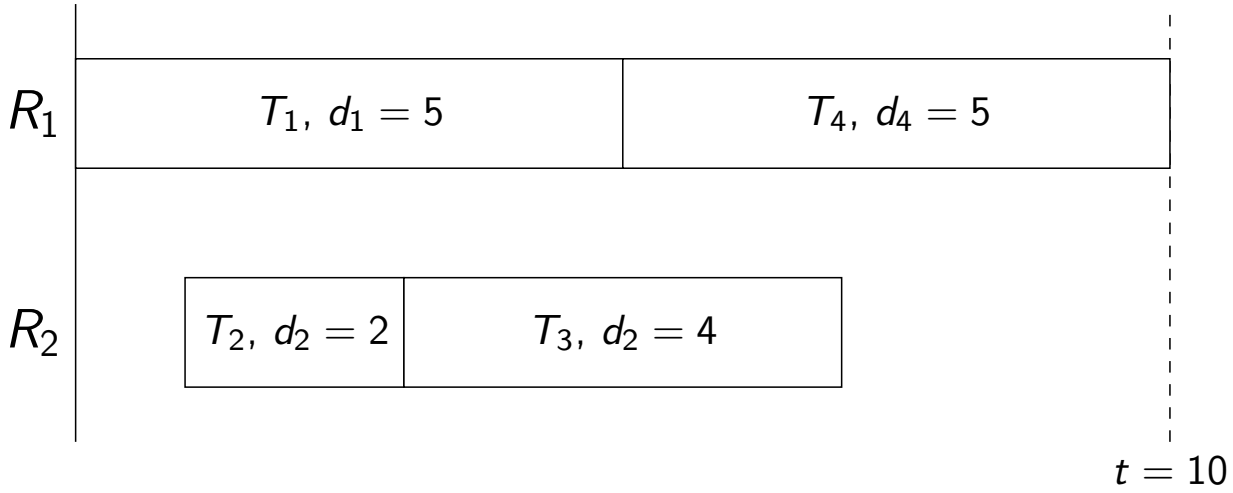
Le coût de l'ignorance

- Deux ressources, des tâches de durées variables qui arrivent à des dates inconnues.
- Algorithme : on lance la tâche arrivée le plus tôt sur la première machine disponible.



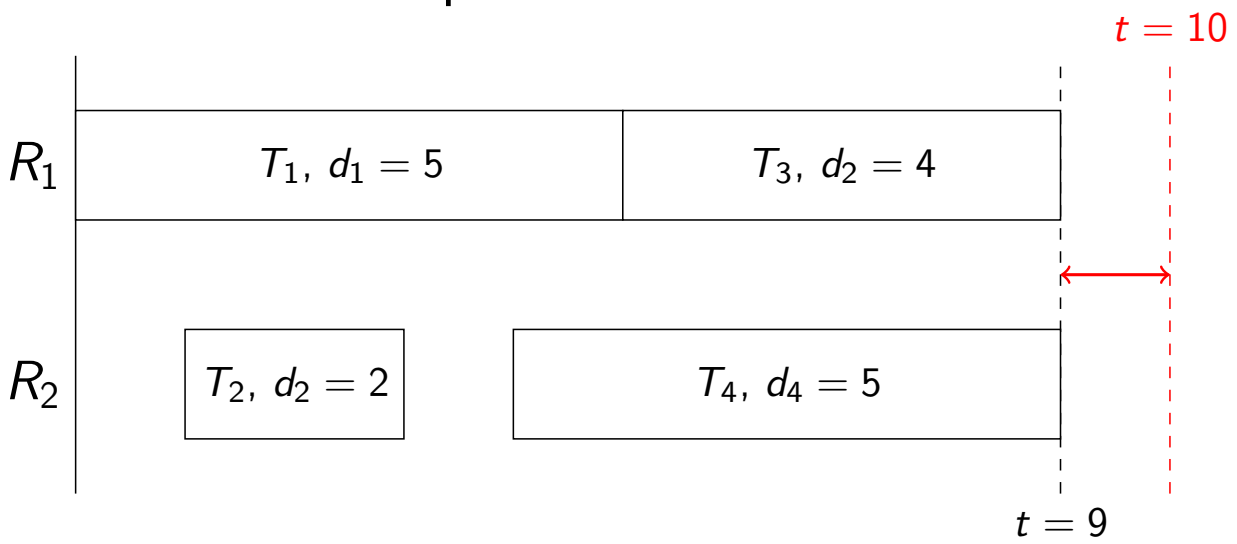
Le coût de l'ignorance

- Deux ressources, des tâches de durées variables qui arrivent à des dates inconnues.
- Algorithme : on lance la tâche arrivée le plus tôt sur la première machine disponible.



Le coût de l'ignorance

- Deux ressources, des tâches de durées variables qui arrivent à des dates inconnues.
- Algorithme : on lance la tâche arrivée le plus tôt sur la première machine disponible.



- Si on avait eu connaissance des arrivées, on aurait pu mieux faire !

Coût de l'ignorance et α-compétitivité

- Le coût de l'ignorance c'est le surcoût (selon la métrique considérée) par rapport à la solution qu'on pourrait avoir en ayant une connaissance parfaite des entrées.

Définition

*Un algorithme online est dit α-**compétitif** si la solution donnée est **au plus** α fois pire que l'optimal (avec connaissance parfaite des données d'entrée).*

- **Ici : on va montrer comment on évalue la compétitivité des algorithmes et comment revoir certaines conceptions.**

Sources

- Présentation fortement inspirée une présentation d'Olivier Beaumont (DR Inria Bordeaux) et de deux publications :
  - *Competitive algorithms for server problems.* Manasse et al. (1990).
  - *Performance Analysis and Optimality Results for Data-Locality Aware Tasks Scheduling with Replicated Inputs.* Beaumont et al. (2019).

- Deux cours sur l'ordonnancement : Loris Marchal (ENS de Lyon) and Peter Brucker (University of Osnabrueck).

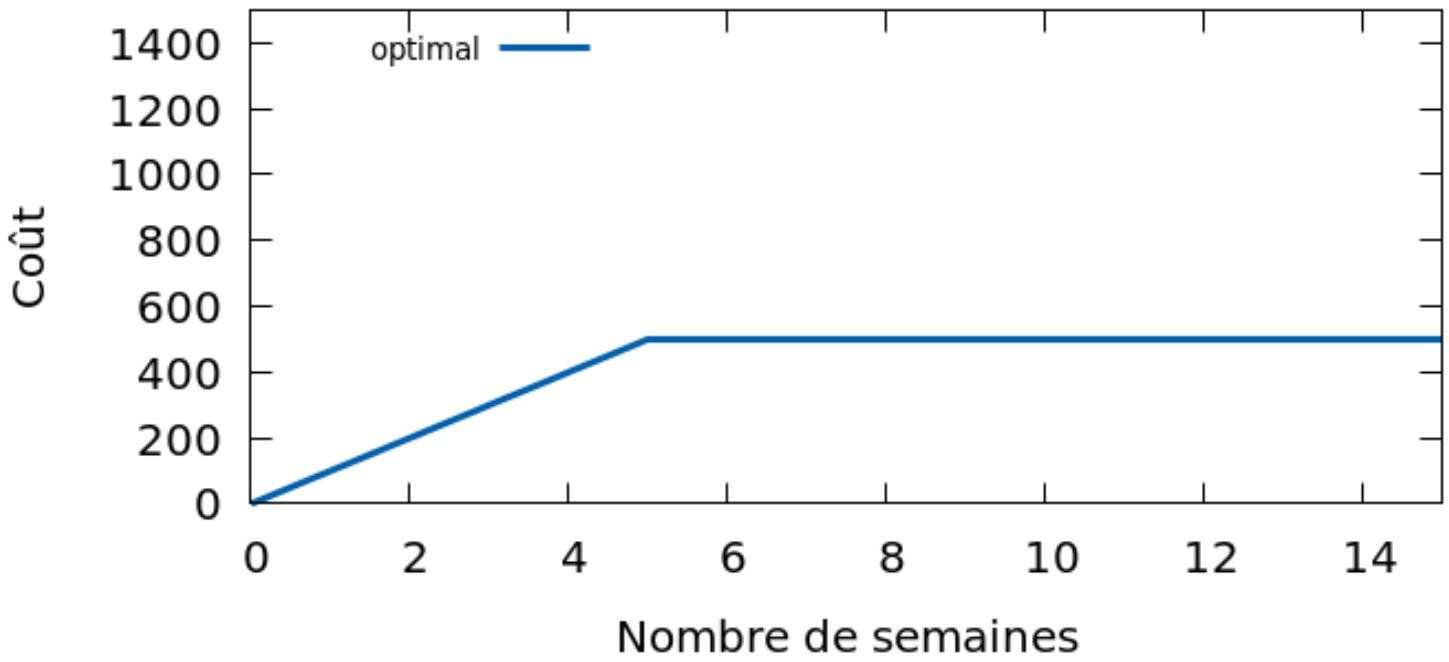
Table des matières

Acheter ou louer des skis ?

- On a deux options :
  - Acheter une paire de skis : 500€ une fois.
  - Louer une paire de skis : 100€/semaine.
- Problème : on ne connaît pas le nombre de semaines où on va skier (sur plusieurs années).
- Question : dois-je acheter ou louer ?
- Problème simplifié : pas d'usure, pas de revente, pas de fluctuation des prix...

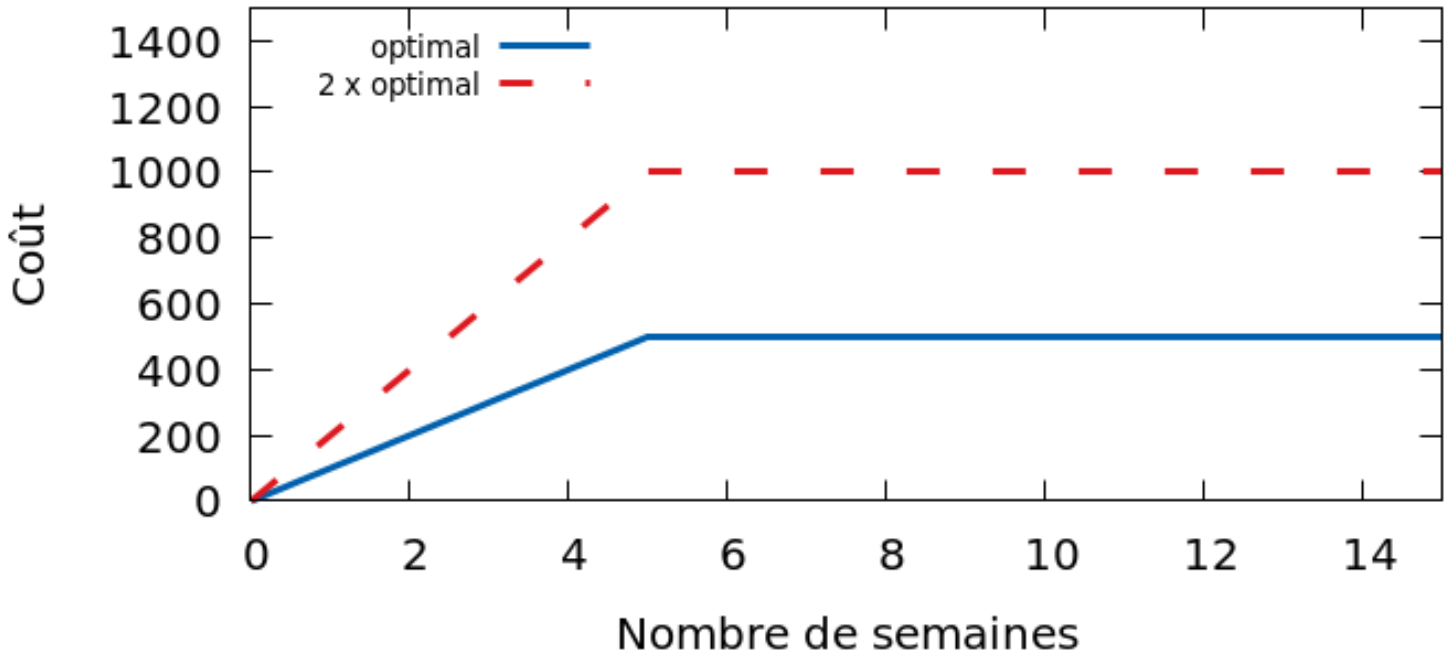
Solution optimale

- Si je connais le nombre de semaines n il est facile de prendre une décision optimale.
  - $n < 5$: on loue : $100 \times n$ €.
  - $n \geq 5$: on achète 500 €.



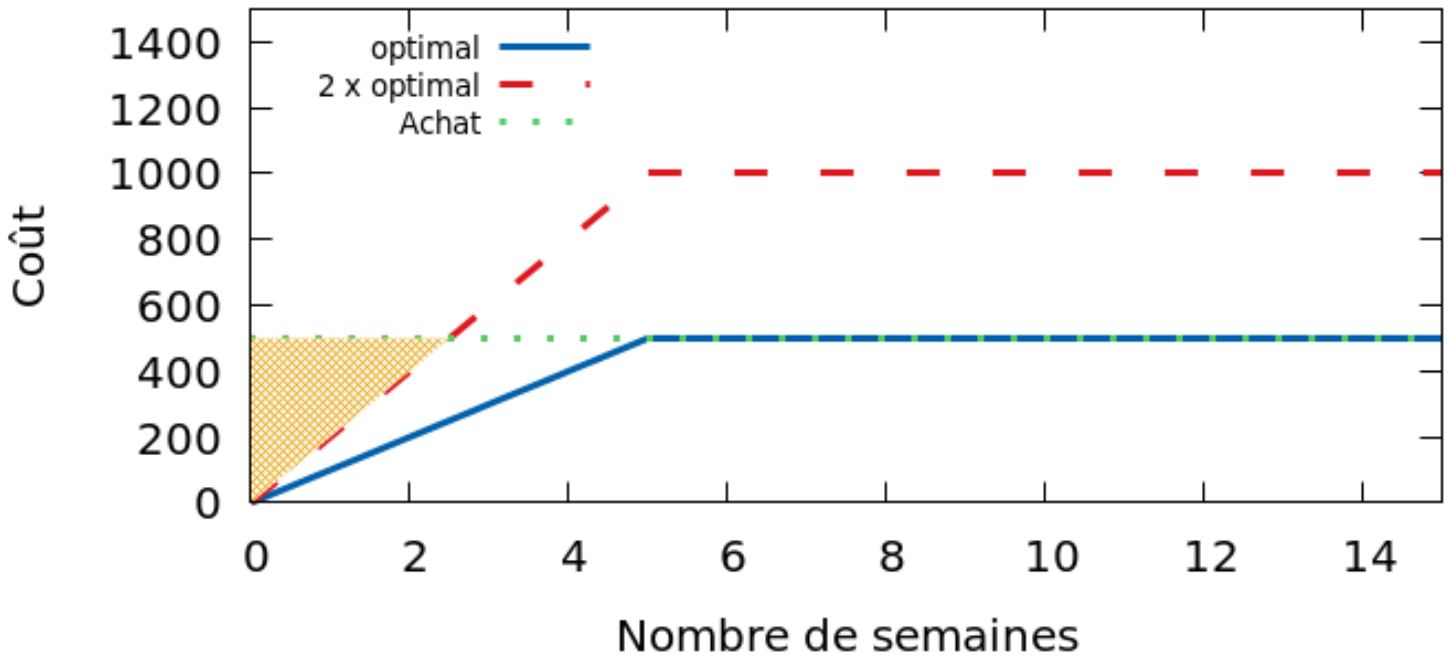
2-Compétitivité

- Objectif : trouver une stratégie qui soit 2-compétitive.



Achat

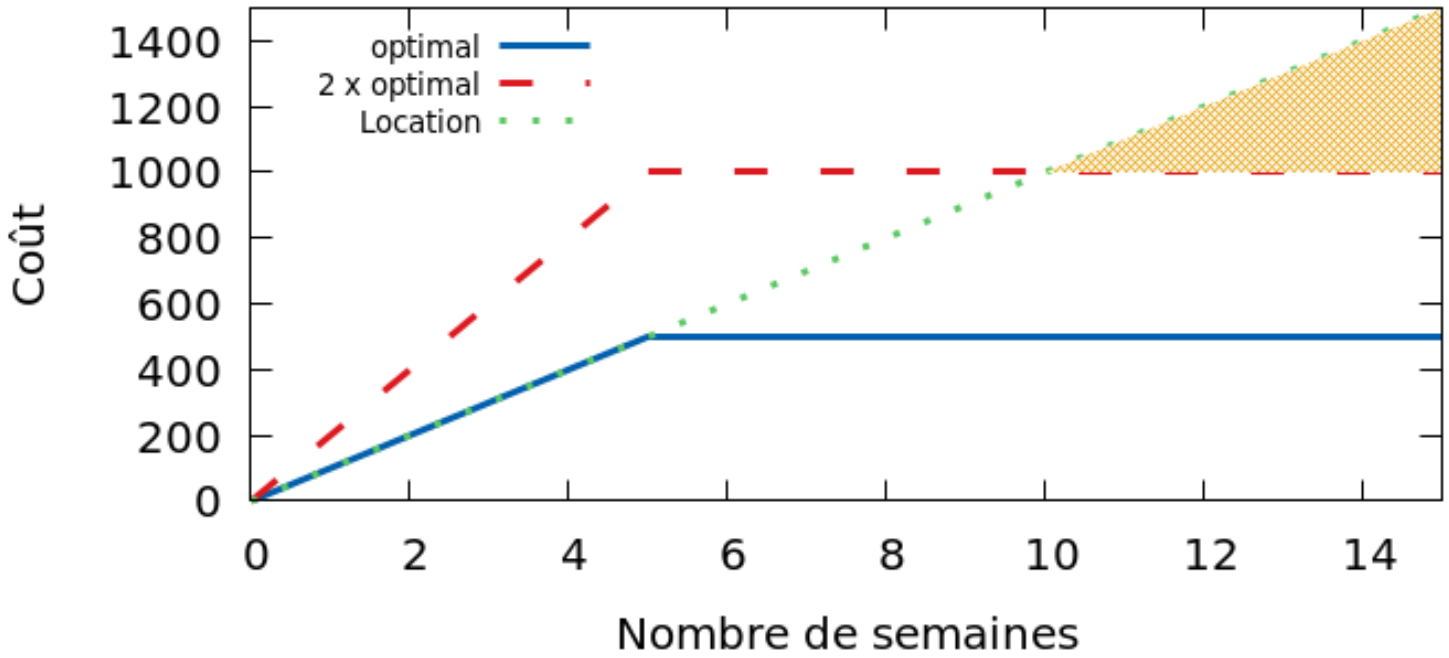
- L'achat immédiat n'est pas 2-compétitif.



- Arbitrairement mauvais : achat/optimal $\underset{n \to 0}{\to} +\infty$.

Location

- La location systématique n'est pas 2-compétitive.



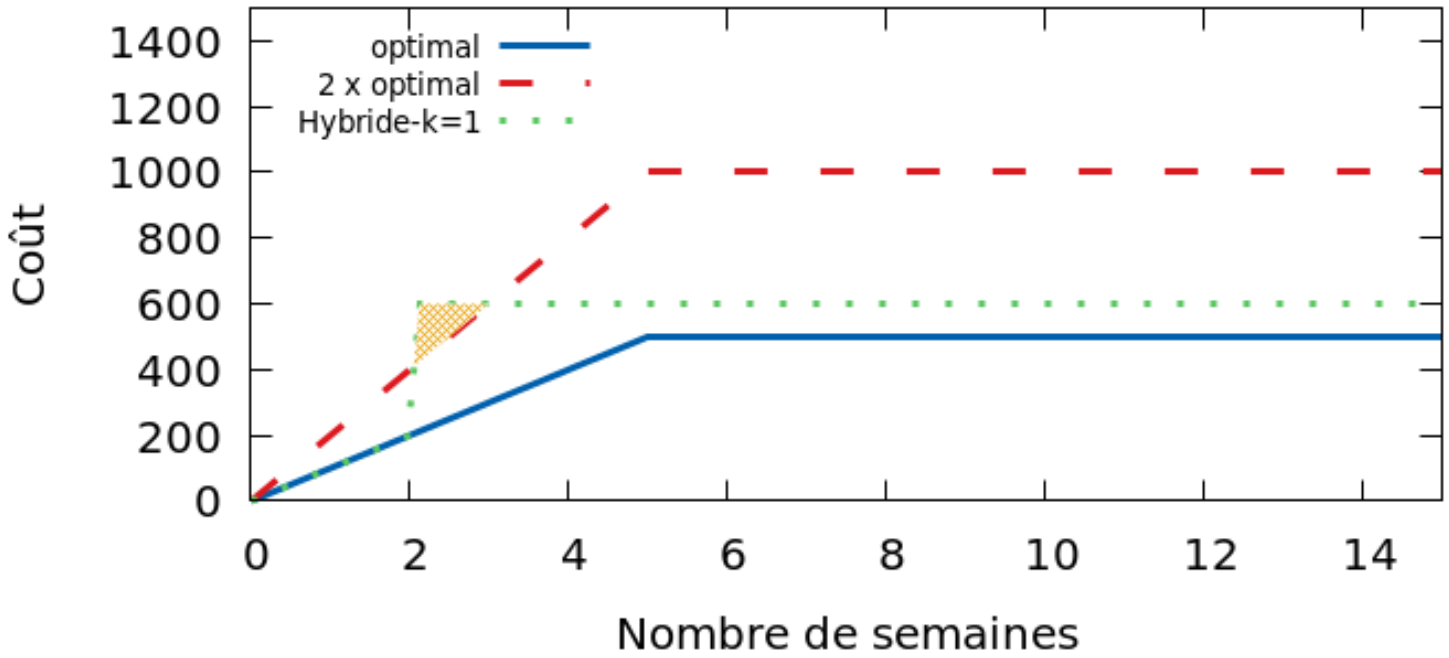
- Arbitrairement mauvais : location/optimal $\underset{n\to+\infty}{\to} +\infty$.

Stratégie hybride

- Acheter dès le début n'est pas satisfaisant.
- Ne passer que par de la location non plus.
- Solution : changer de stratégie au milieu :
  - Les k premières semaines on loue.
  - A partir de $k+1$ on achète.
- k est fixé pour la stratégie.
- Coût :
  - Si $n \leq k$: $100 \times n$ €.
  - Si $n > k$: $500 + 100 \times k$ €.

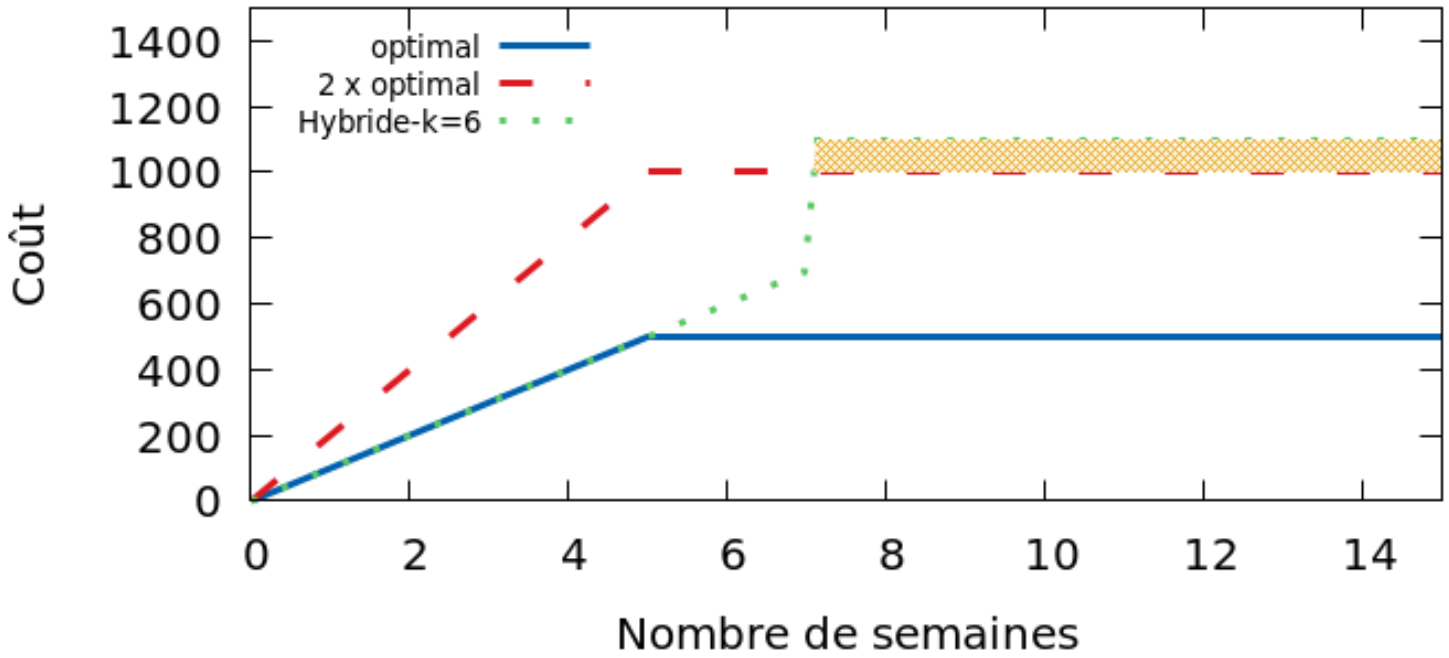
Stratégie Hybride

- $k = 1, 2 \rightarrow$ trop petit !



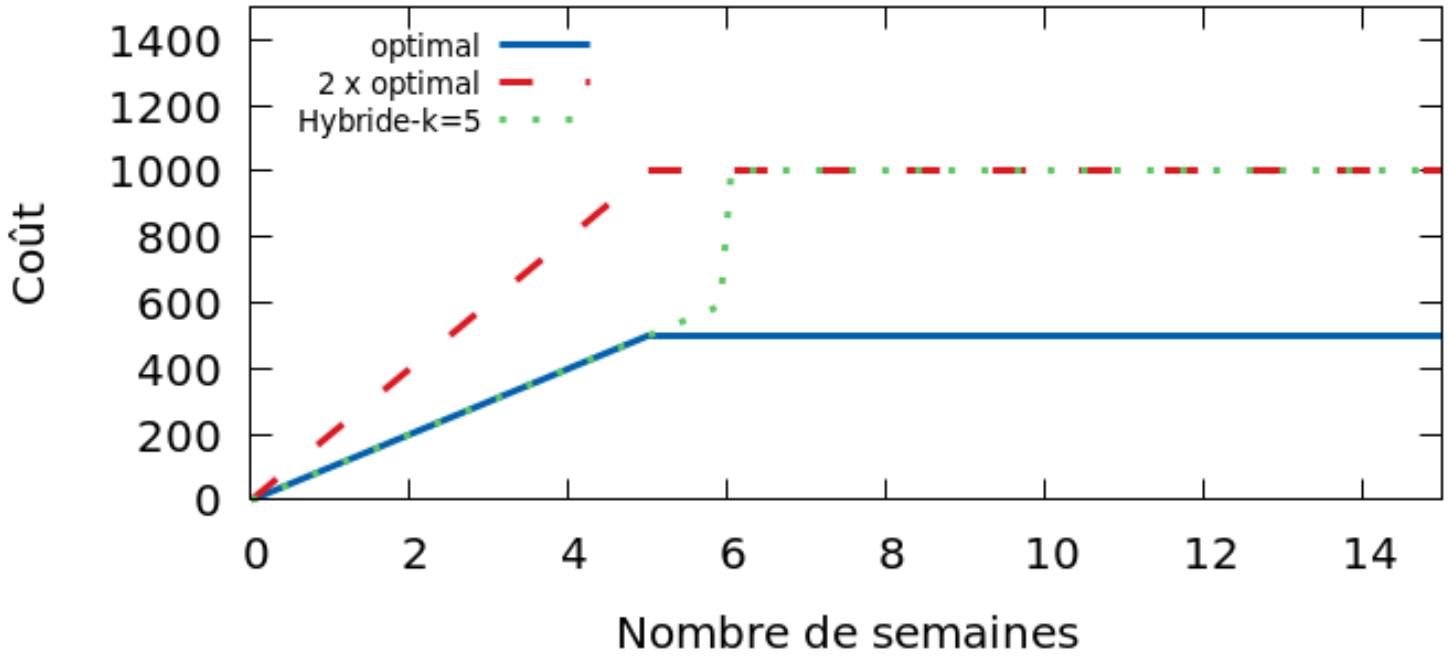
Stratégie Hybride

- $k \geq 6 \rightarrow$ trop grand !



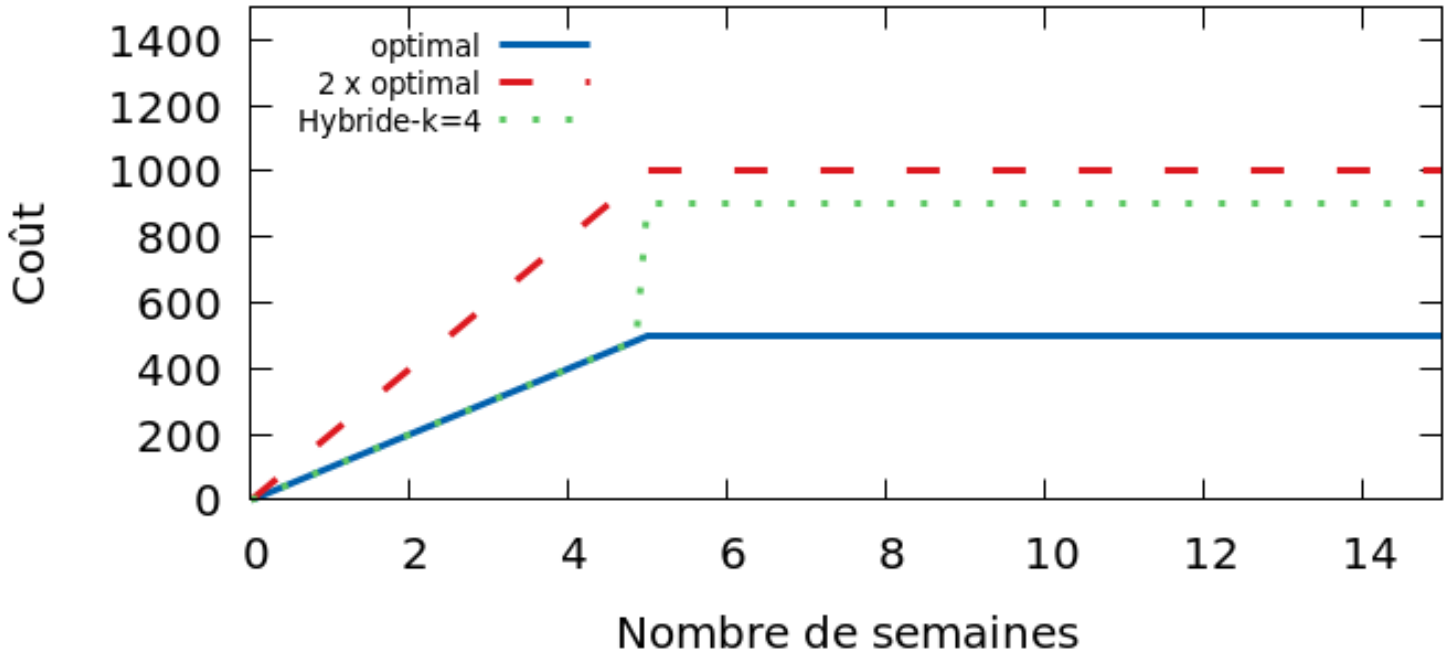
Stratégie Hybride

- $k = 5 \rightarrow$ exactement 2-compétitif (pareil pour 3) !



Stratégie Hybride

- $k = 4 \rightarrow$ 1.8-compétitif (optimal) !



Conclusion

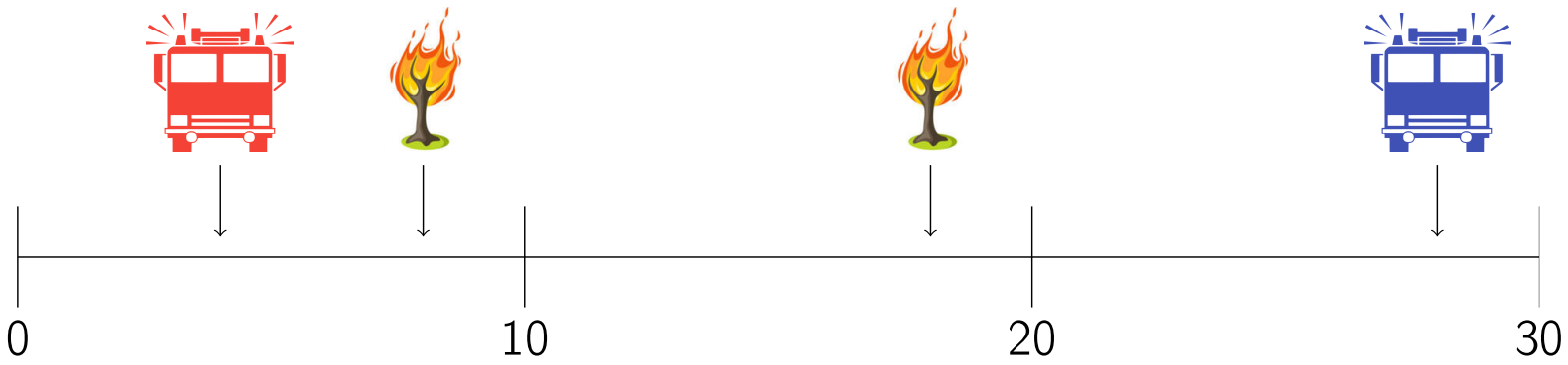
- Ici il faut savoir changer de stratégie au bon moment et ne pas trop anticiper.
- On peut généraliser le résultat précédent pour d'autres prix d'achat ou de location.
  - Il existera toujours une stratégie 2-compétitive.
  - k dépendra du ratio achat/location.

- Applications pratiques :
  - maintenance (réparation vs remplacement).
  - Dimensionnement de ressources (serveurs, Cloud Computing).

Table des matières

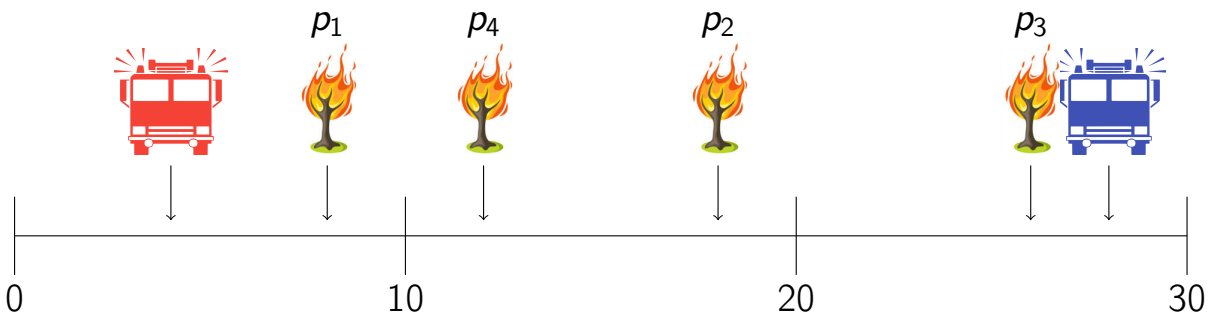
Présentation du problème

- Surveillance d'une route de 30km dans les landes :
  - Droite, uniforme, et qui prend feu souvent.
- On dispose de deux camions.
- On ne considérera pas le temps de maîtrise de l'incendie et on supposera que l'on détecte instantanément les feux.
- Métrique : minimiser la distance parcourue par les camions.



Solution optimale

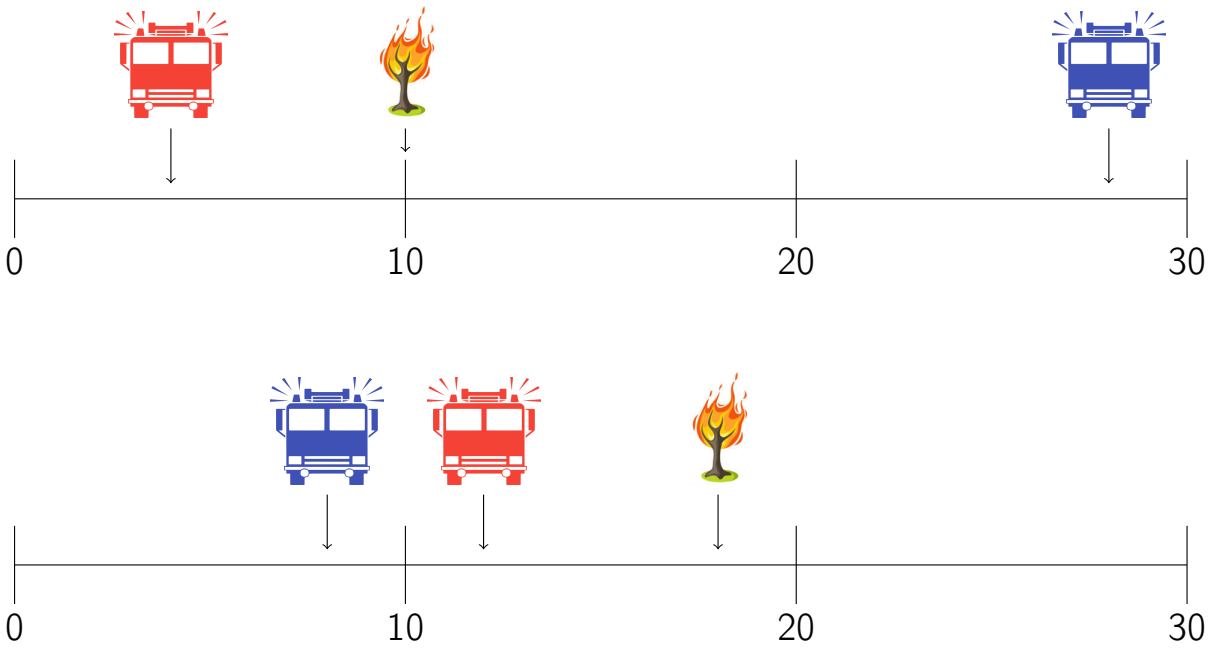
- Si on connaît les positions des feux et l'ordre d'apparition (p_1, $p_2 \ldots$) on peut calculer une solution optimale.
- Mais c'est beaucoup plus difficile que dans le problème précédent !



- Programmation dynamique :
  - $CoutOpt(k, x, y) = \min_{x', y'}(CoutOpt(k-1, x', y) + d(x', p_k), CoutOpt(k-1, x, y') + d(y', p_k))$
  - x, x', y, y' positions (nombre fini de cibles possibles).
  - k nombre de feux.

Construction d'une stratégie online

- Solution optimale beaucoup plus compliquée → difficile de se baser dessus.
- Situation générale : Positions des deux camions et position du feu qui vient de se déclarer.



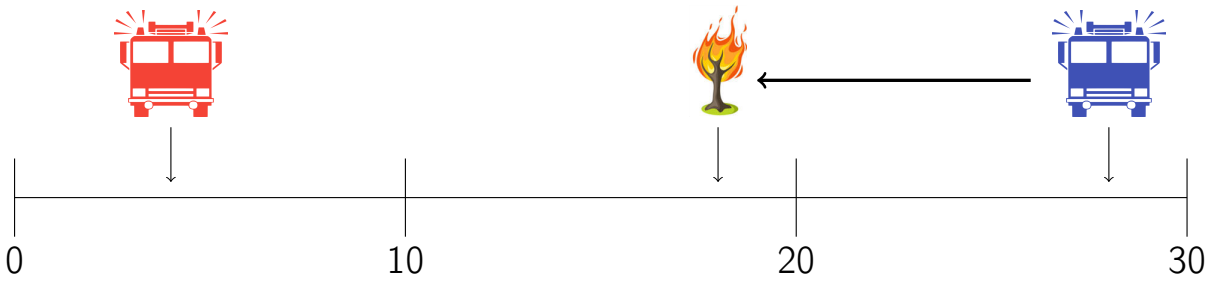
Solution gloutonne

Glouton : Le camion le plus proche va éteindre le feu.

- Optimal pour chaque feu.
- Et sur l'ensemble des feux ?
  $\hookrightarrow$ Il peut être très mauvais !
- Trouvons un cas "simple" où l'optimal est facile à calculer.

Solution gloutonne

Glouton : Le camion le plus proche va éteindre le feu.



- Optimal pour chaque feu.
- Et sur l'ensemble des feux ?
  $\hookrightarrow$ Il peut être très mauvais !
- Trouvons un cas "simple" où l'optimal est facile à calculer.

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

- On reproduit n fois la séquence suivante.

- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

- On reproduit n fois la séquence suivante.

- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

- On reproduit n fois la séquence suivante.

- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

- On reproduit n fois la séquence suivante.

- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

- On reproduit n fois la séquence suivante.

- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

- On reproduit n fois la séquence suivante.

- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

- On reproduit n fois la séquence suivante.

- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

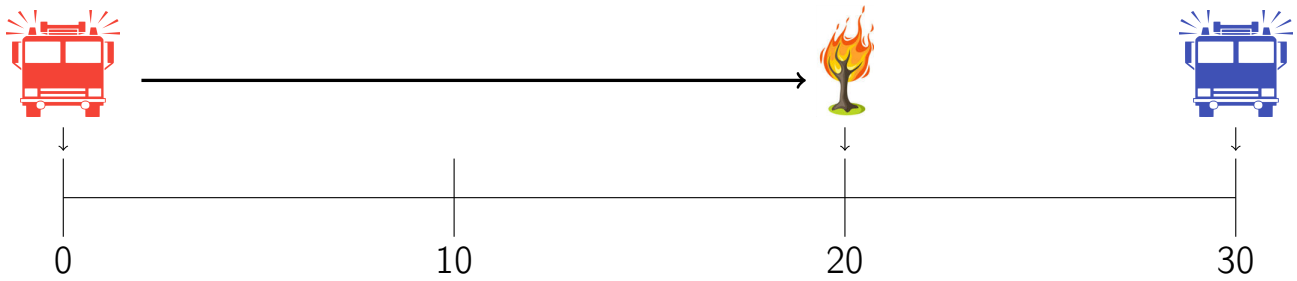
- On reproduit n fois la séquence suivante.

- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

- On reproduit n fois la séquence suivante.

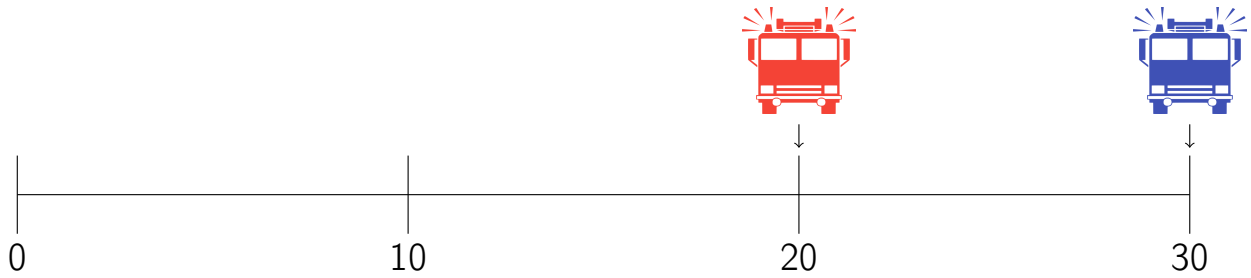


- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Solution gloutonne

Technique de l'adversaire : il connaît notre algorithme et choisit la position des feux en fonction.

- On reproduit n fois la séquence suivante.

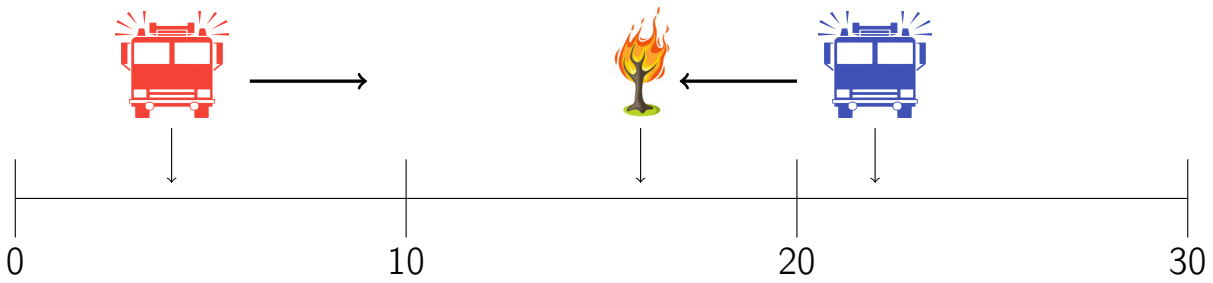


- Coût avec Glouton : $20 \times n$.
- Coût optimal : 20.
- Comment faire mieux ?
- A quel point peut-on faire mieux ?

Une nouvelle approche "Balance"

- Jouer l'optimal à chaque coup (Glouton) ne marche pas.
- On va introduire des mouvements inutiles.

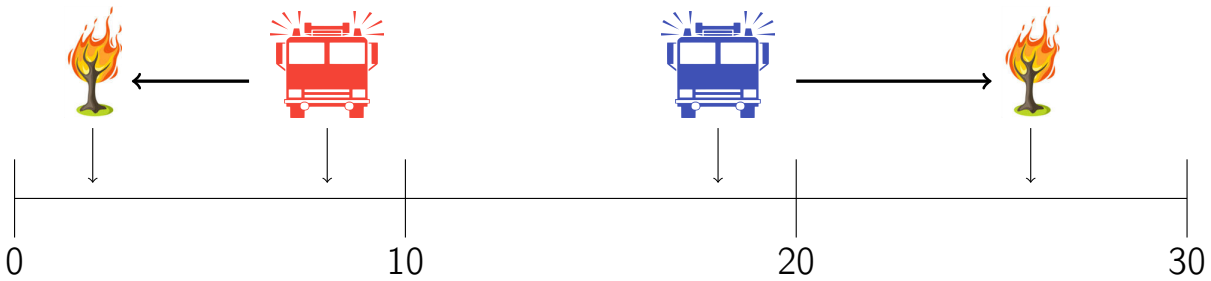
Balance : si le feu est au milieu on bouge les deux camions de la même distance. Sinon on bouge le plus proche.



Une nouvelle approche "Balance"

- Jouer l'optimal à chaque coup (Glouton) ne marche pas.
- On va introduire des mouvements inutiles.

Balance : si le feu est au milieu on bouge les deux camions de la même distance. Sinon on bouge le plus proche.



Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 0.
- Coût optimal : 0.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 0.
- Coût optimal : 0.

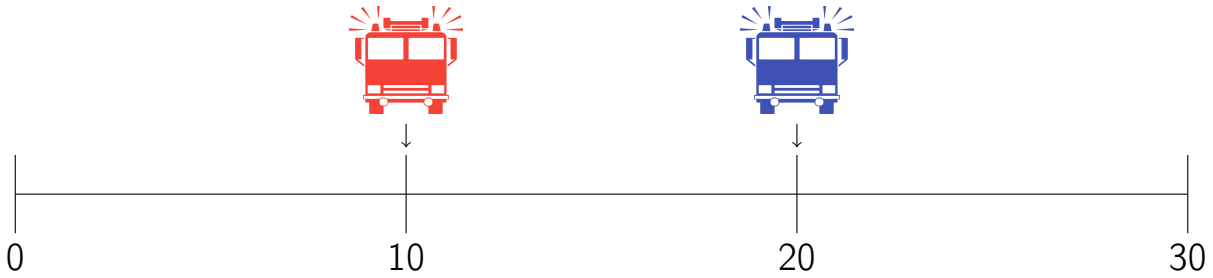
Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 0.
- Coût optimal : 0.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :



- Coût avec Balance : 20.
- Coût optimal : 10.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 20.
- Coût optimal : 10.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 20.
- Coût optimal : 10.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 30.
- Coût optimal : 20.

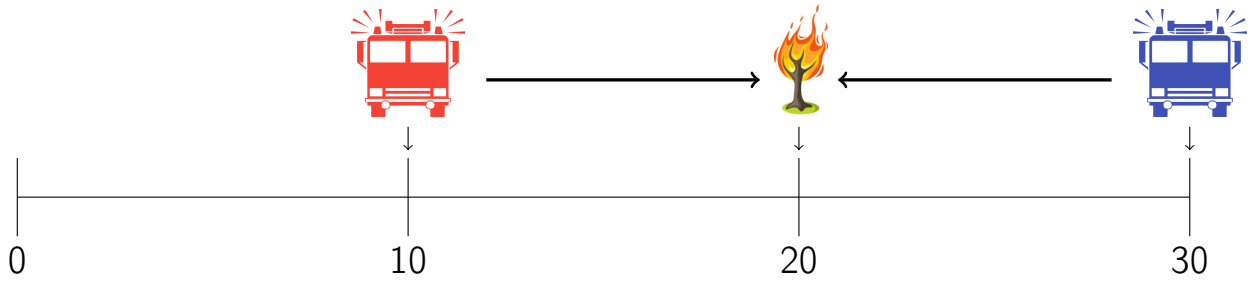
Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 30.
- Coût optimal : 20.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :



- Coût avec Balance : 30.
- Coût optimal : 20.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 50.
- Coût optimal : 20.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 50.
- Coût optimal : 20.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

- Coût avec Balance : 50.
- Coût optimal : 20.

Compétitivité

- Reprenons l'exemple qui a mis Glouton en échec :

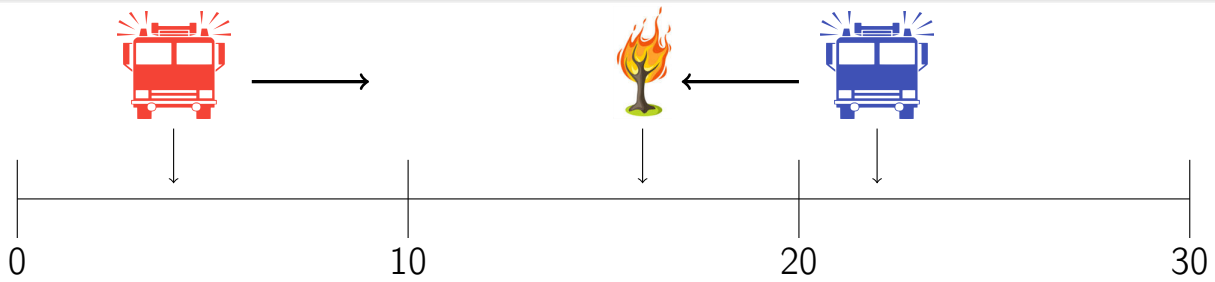
- Coût avec Balance : 60.
- Coût optimal : 20.

Balance est 3-compétitif.

Balance paresseux

- Avec un peu de mémoire on peut faire mieux !
- Paresseux : *qui est enclin à éviter l'action, le travail, l'effort, à ne pas se donner de peine.*
- Paresseux (en informatique) : *programme vertueux qui ne fait que le travail nécessaire.*

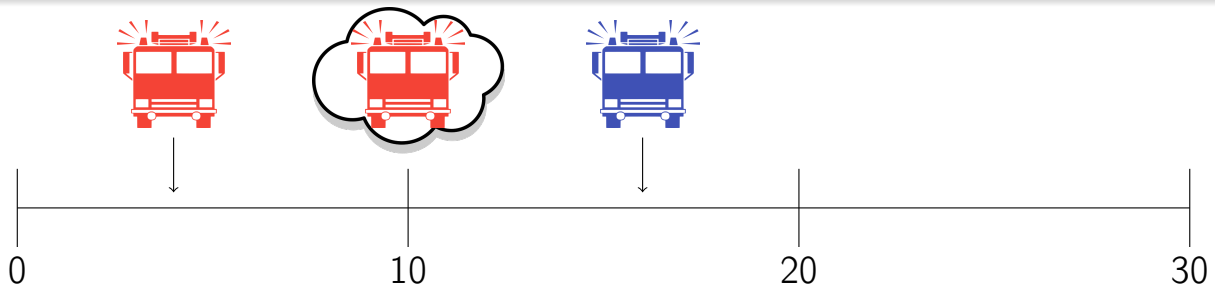
Balance paresseux : Balance, mais si les deux camions doivent bouger, seul le pus proche se déplace. Le second enregistre une position imaginaire. En cas d'égalité, on bouge le camion qui a le moins bougé.



Balance paresseux

- Avec un peu de mémoire on peut faire mieux !
- Paresseux : *qui est enclin à éviter l'action, le travail, l'effort, à ne pas se donner de peine.*
- Paresseux (en informatique) : *programme vertueux qui ne fait que le travail nécessaire.*

Balance paresseux : Balance, mais si les deux camions doivent bouger, seul le pus proche se déplace. Le second enregistre une position imaginaire. En cas d'égalité, on bouge le camion qui a le moins bougé.



Compétitivité

- Retour encore au même exemple !

- Coût avec Balance paresseux : 0.
- Coût optimal : 0.

Compétitivité

- Retour encore au même exemple !

- Coût avec Balance paresseux : 0.
- Coût optimal : 0.

Compétitivité

- Retour encore au même exemple !

- Coût avec Balance paresseux : 0.
- Coût optimal : 0.

Compétitivité

- Retour encore au même exemple !

- Coût avec Balance paresseux : 10.
- Coût optimal : 10.

Compétitivité

- Retour encore au même exemple !

- Coût avec Balance paresseux : 10.
- Coût optimal : 10.

Compétitivité

- Retour encore au même exemple !

- Coût avec Balance paresseux : 10.
- Coût optimal : 10.

Compétitivité

- Retour encore au même exemple !

- Coût avec Balance paresseux : 20.
- Coût optimal : 20.

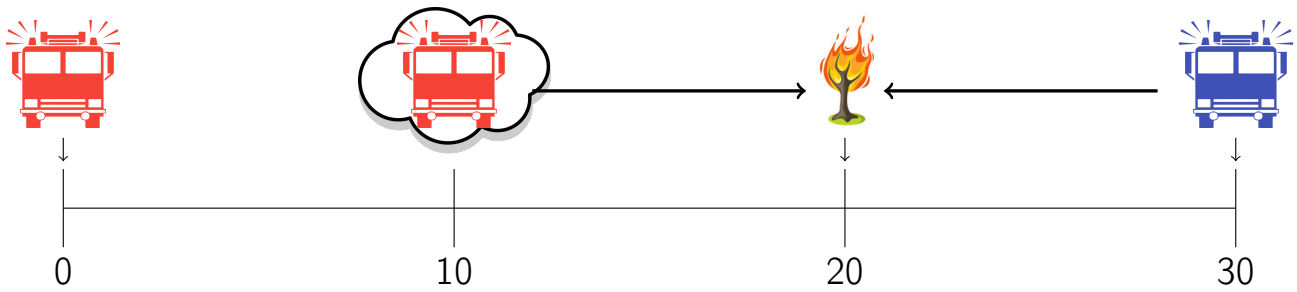
Compétitivité

- Retour encore au même exemple !

- Coût avec Balance paresseux : 20.
- Coût optimal : 20.

Compétitivité

- Retour encore au même exemple !



- Coût avec Balance paresseux : 20.
- Coût optimal : 20.

Compétitivité

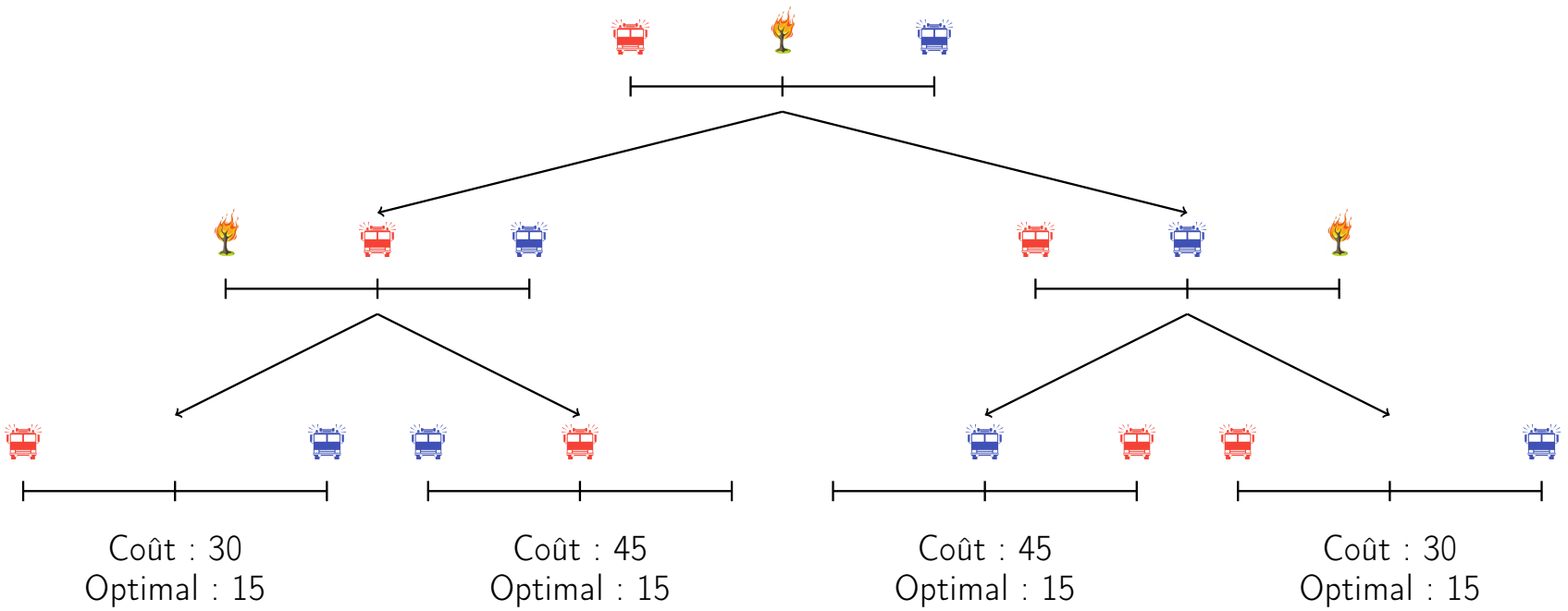
- Retour encore au même exemple !

- Coût avec Balance paresseux : 40.
- Coût optimal : 20.

Balance paresseux est 2-compétitif.

Coût de l'ignorance

- Peut-on faire mieux que 2-compétitif ?
- Essayons d'explorer de manière exhaustive un cas simple (deux feux choisis par un adversaire).



Il n'y a pas de stratégie α-compétitive avec $\alpha < 2$.

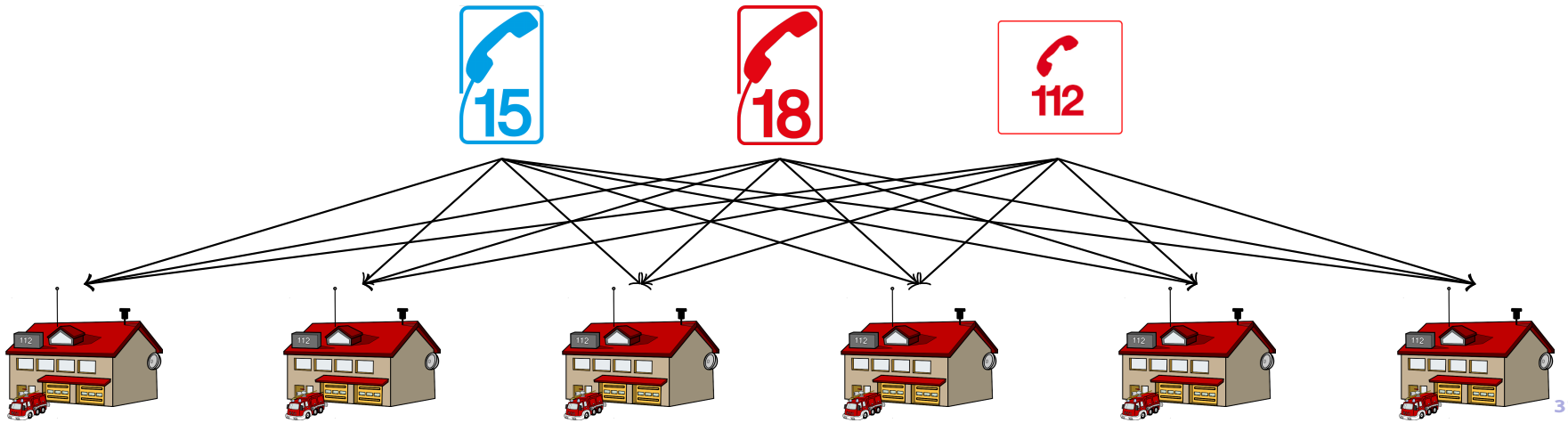
Conclusion

- Jouer chaque coup comme si c'était le dernier ça ne marche pas.
- Il faut parfois accepter des mouvements inutiles.
  $\hookrightarrow$ C'est encore mieux si on arrive à en éviter une partie.
- Extensions :
  - Balance paresseux est k-compétitif si on a k camions.
  - Des versions aléatoires sans mémoires existent aussi.
- Applications :
  - Déplacement de têtes de lecture de disques.
  - Requêtes sur des serveurs.

Table des matières

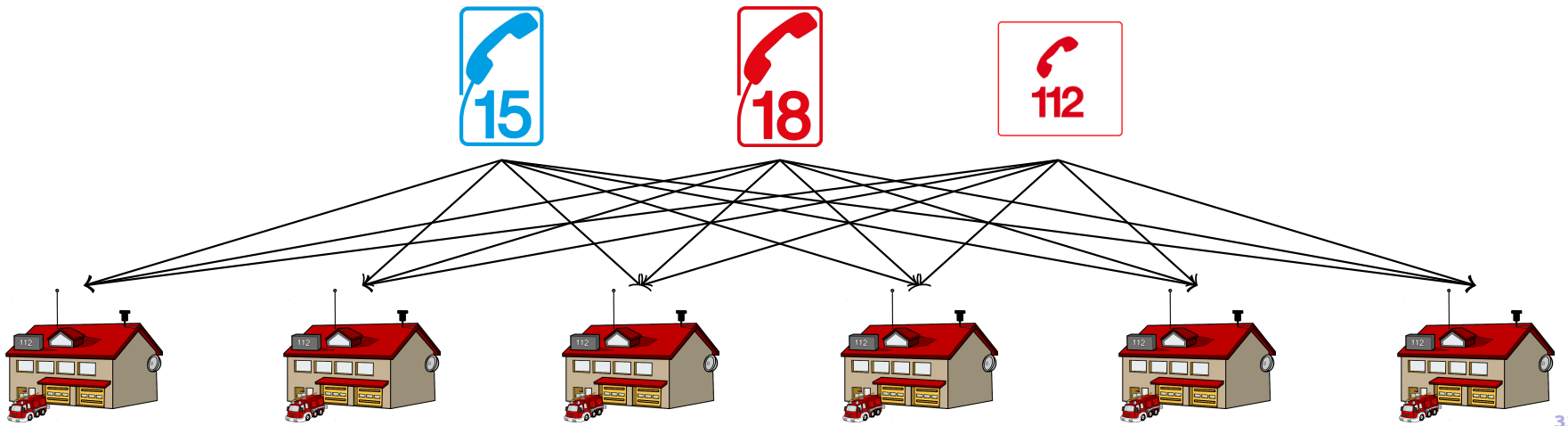
Un autre type d'incertitude

- Dans les deux exemple précédents on ne connaissait pas le nombre d'entrées à l'avance.
- Ici on rajoute une connaissance seulement partielle de ces entrées.
  - Systèmes distribués.
- Exemple : répartition de demande d'intervention sur plusieurs casernes.
  - Plusieurs sources de demandes. Pas de gestion centralisée.



Présentation du problème

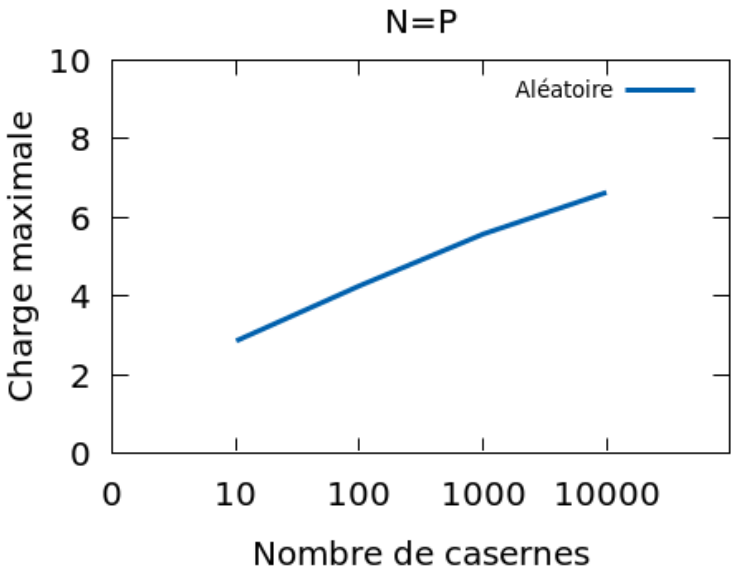
- L'opérateur doit choisir la caserne à qui envoyer la demande.
- S'il ne demande pas aux casernes, il n'a pas connaissance de leurs charges respectives.
  - Demander à toutes les casernes prend trop de temps et risque de saturer le réseau.
- Objectif : minimiser la charge maximum des casernes.
- On va définir une stratégie utilisée par tous les opérateurs.

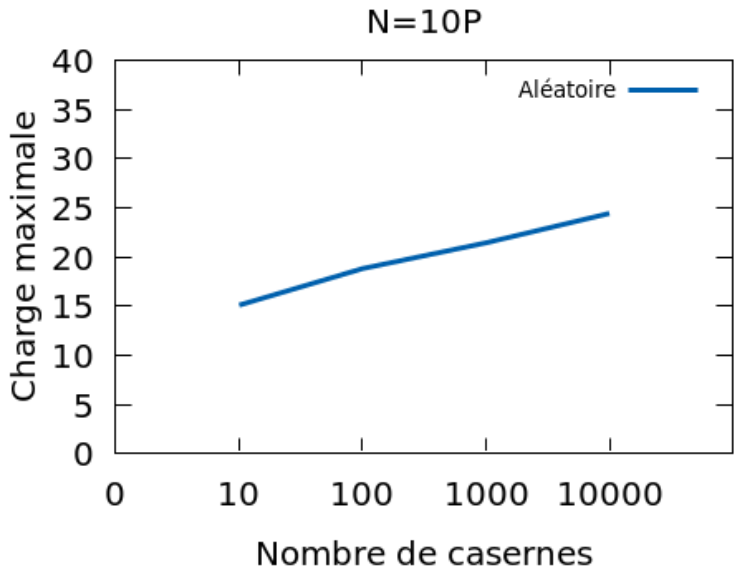


Stratégie aléatoire

Aléatoire : On tire une caserne aléatoire (tirage uniforme).

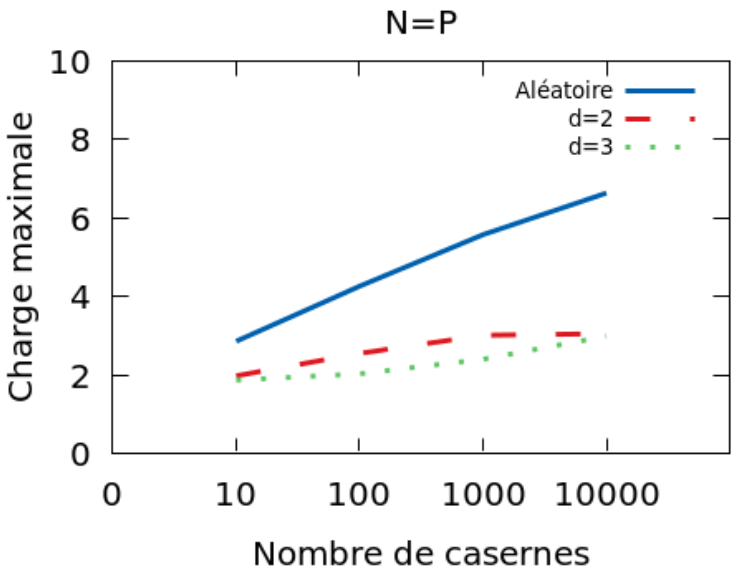
- Pas de prises d'information.
- A quel point est-ce bon ?
- Simulations : 100 essais pour P casernes et $N = P$ et $N = 10 \times P$ appels.

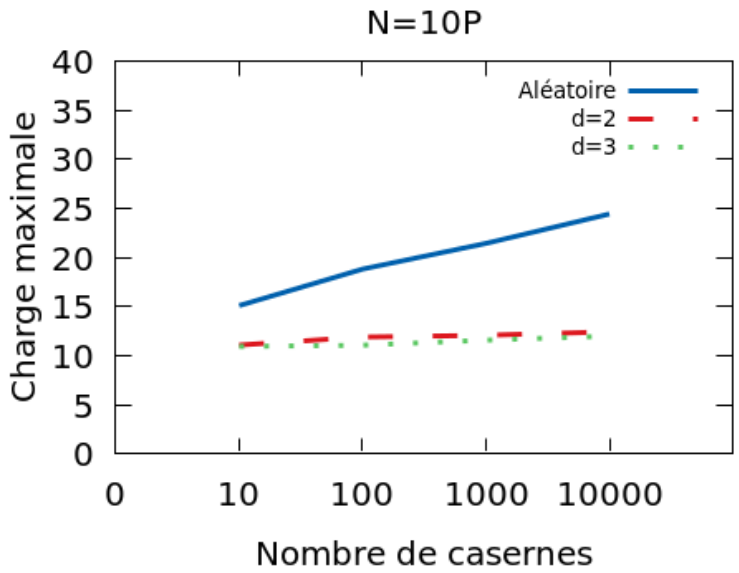




Choix multiples

- Essayons d'utiliser un peu d'informations.
- Appelons d casernes avant de prendre une décision :
  - On donne l'intervention à la caserne la moins chargée.

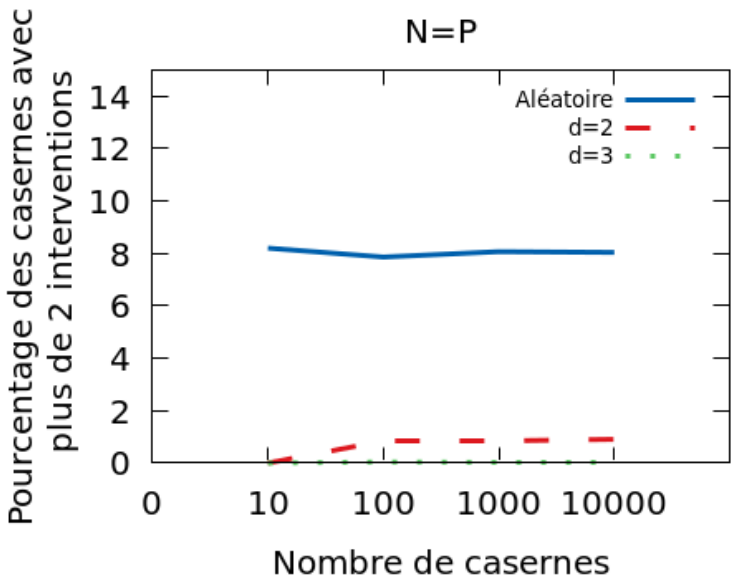


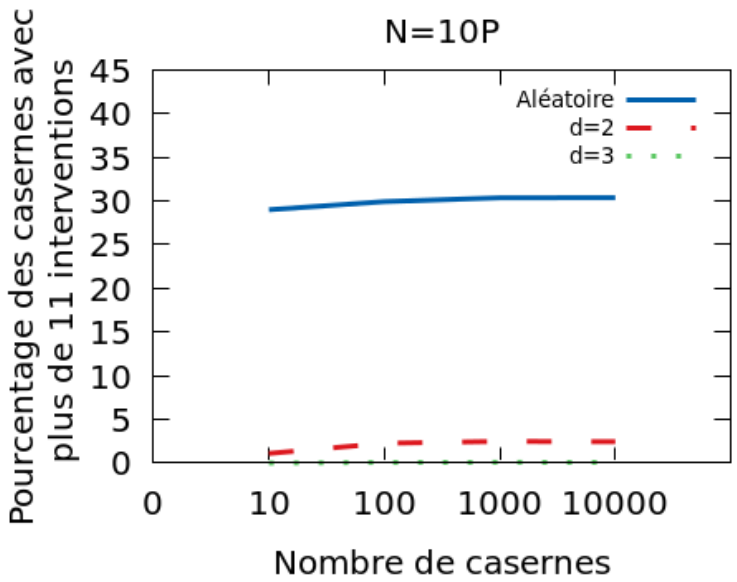


- Se donner une caserne supplémentaire améliore grandement la répartition des interventions.

Choix multiples

- Essayons d'utiliser un peu d'informations.
- Appelons d casernes avant de prendre une décision :
  - On donne l'intervention à la caserne la moins chargée.



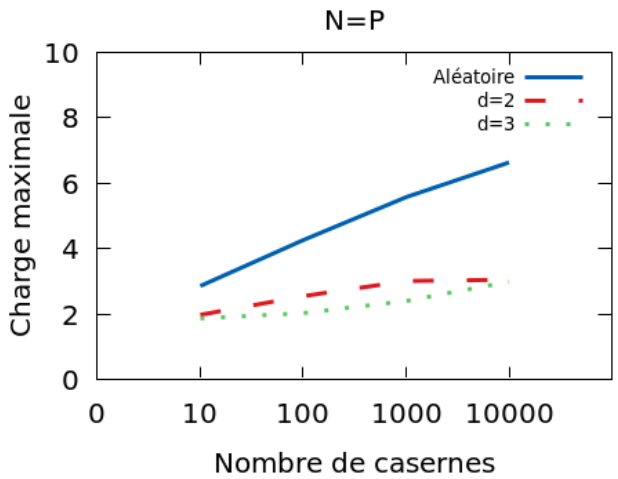


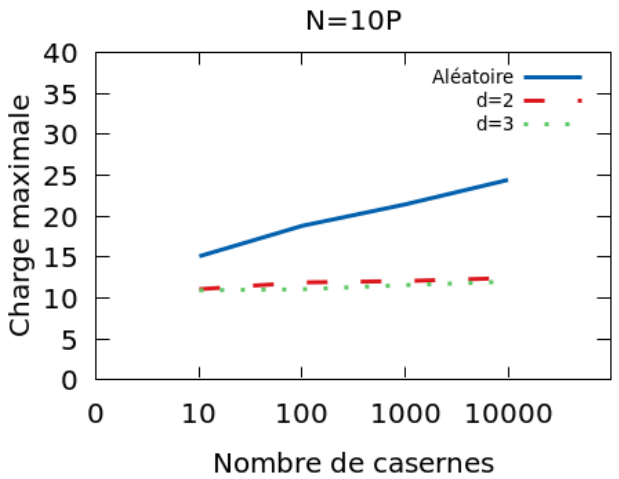
- Se donner une caserne supplémentaire améliore grandement la répartition des interventions.

Un peu de théorie

- Études probabilistes.
- On peut évaluer l'espérance de la charge maximum.

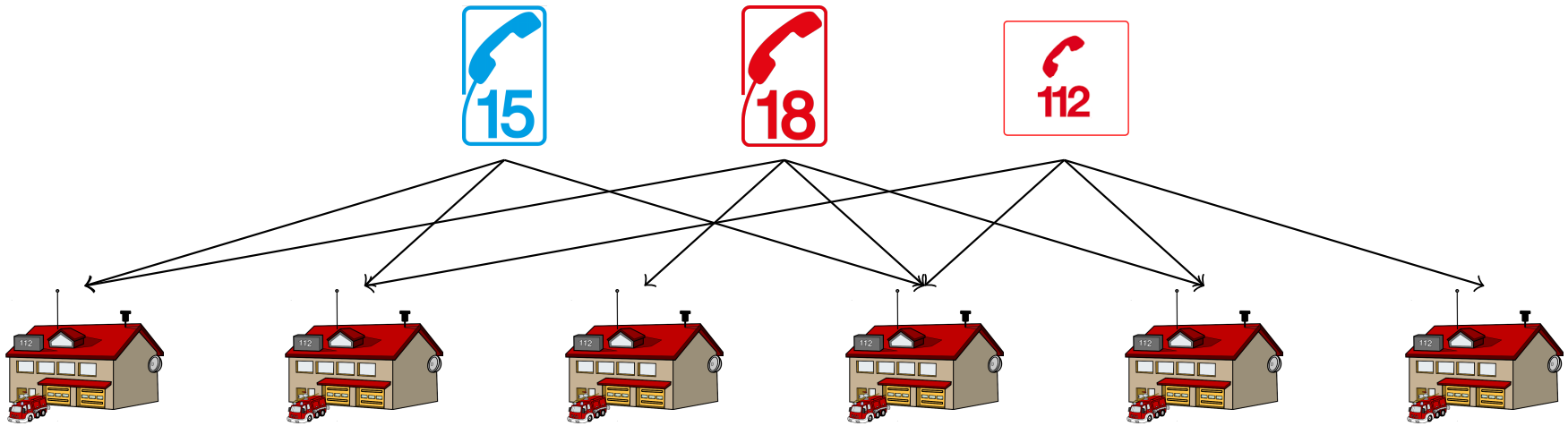
	Aléatoire (Raab and Martin, 1998)	d-choix ($d \geq 2$) (Berenbrink et al. 2003)
$N = M$	$O\left(\frac{\log N}{\log \log N}\right)$	$1 + O(\log \log M)$
$N = 10M$	$\frac{N}{M} + O(\sqrt{\log M})$	$\frac{N}{M} + O(\log \log M)$





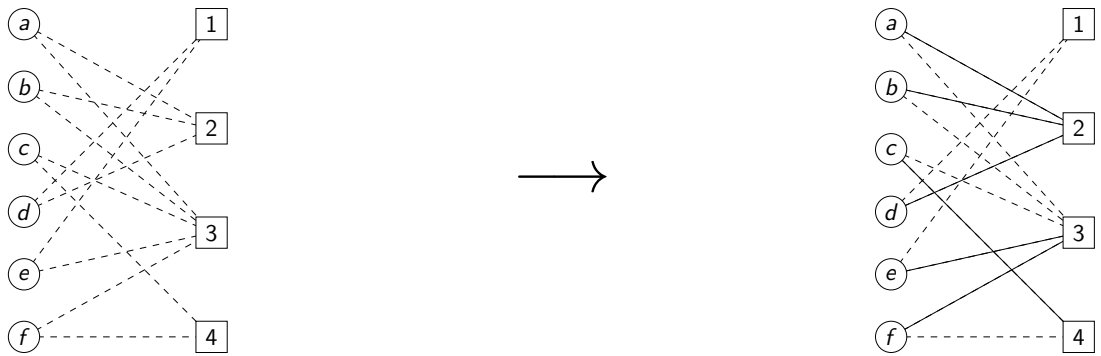
Extensions

- Les résultats s'étendent :
  - au cas où les interventions n'ont pas la même durée.
  - au cas où chaque opérateur n'a accès qu'à un sous-ensemble des casernes.



Map/Reduce

- On peut aussi retourner le problème :
  - Les opérateurs donnent les interventions à toutes les casernes qui choisissent ensuite au hasard une interventions non effectuée.
  - Exemple d'application : Map/Reduce (Hadoop, Spark).
  - distributions de tâches dépendant de fichiers répliqués.



- Si chaque fichier est répliqué au moins deux fois, la machine la plus chargée fera $\frac{N}{M} + O(\log \log M)$ tâches.

Conclusion

- Ici une autre méthode d'analyse que la compétitivité (plus difficile à définir en distribué).
- Parfois il suffit de peu d'information pour améliorer significativement une solution basique.
- Applications : distribution de requêtes.

Table des matières

Résumé

- L'algorithmique classique se base sur connaissance parfaite des entrées et de son environnement.
- En pratique, ce n'est pas toujours applicable.
- Cependant, on peut quand même produire des solutions efficaces et les évaluer.
- Ici : l'incertitude venait du nombre d'événements.
  - Il faut savoir faire évoluer sa stratégie.
  - Viser l'optimal à chaque étape n'est pas toujours une bonne idée.
  - On gagne quand même souvent à avoir un peu plus d'information.

Ce n'était qu'un aperçu

- Avec les systèmes à large échelle ou distribués il devient de plus en plus difficiles de rester dans le modèle "offline".
  - Nombre d'évènements et état des autres machines (entraperçu ici).
  - Caractéristiques de tâches (durées d'exécution, par exemple).
  - Pannes, fautes et ralentissements !
    - Très peu d'erreurs sur un processeur isolé (une toute les 125 ans en moyenne).
    - A grande échelle (50 000 processeurs) c'est une par jour !
- Beaucoup, beaucoup de techniques proposées :
  - Algorithmes onlines, réplication, checkpoint...
  - Compétitivité, modèles probabilistes, simulations...