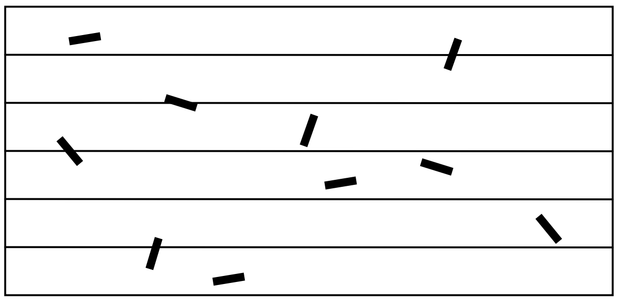
Chapitre 10
Algorithmes probabilistes

- Jouent à pile ou face
- Se comportent différemment lorsque exécutés deux fois sur un même exemplaire
- Défient parfois l'intuition

Première surprise:
le hasard peut être utile!

- Créer le genre humain (???)
- Estimer π



- Cryptographie
- Vérifier la primalité
- Accélérer une recherche
- Réduire l'effet de mauvais exemplaires

Deuxième surprise: le hasard peut être précis

Exemple: pile =succès face =échec

Prob[succès] = ½

Prob[succès après *2* essais] =¾

…

Prob[succès après n essais] = $1 - (½)^n$

n =1000 $\Rightarrow$ échec moins probable qu'une erreur interne de l'ordinateur après une seconde de calcul!

3 types d'algos probabilistes

- Numérique: retourne une solution approximative à un problème numérique (ex: la simulation). Plus de temps $\Rightarrow$ plus grande précision.

- Monte Carlo: retourne toujours réponse (ex: oui ou non) mais peut se tromper. Plus de temps $\Rightarrow$ plus grande probabilité que la réponse soit bonne. En général, on **ne peut pas** déterminer efficacement si réponse obtenue bonne.

- Las Vegas: ne retourne jamais une réponse inexacte, mais parfois ne trouve pas de réponse du tout! Plus de temps $\Rightarrow$ plus grande probabilité de succès sur **chaque** exemplaire. **Il ne s'agit pas** seulement d'être malchanceux sur quelques exemplaires catastrophiques.

Quand Colomb a découvert l'Amérique?

- Numérique: Au 15ième siècle,
  entre 1485 et 1495,
  entre 1493 et 1494,
  …

- Monte Carlo: 1492, 1501, 1492, 1492,
  1487, 555, 1487, … , …

- Las Vegas: 1492, 1492, sais pas, 1492,
  sais pas, 1492, … , …

10.3 Temps espéré ≠ temps moyen

- Moyen(n) = $$\frac{\Sigma_{|w|=n} \text{temps sur exemplaire } w}{\#\{w : |w| = n\}}$$

- Espéré: défini sur chaque exemplaire w

$$\text{Espoir}(w) = \sum_{\textit{choix de l'algo}} \text{temps}(w \text{ avec } \textit{choix}) \times \Pr[\textit{choix}]$$

- Espéré(n) = $\text{Max}_{|w|=n} \{\text{Espoir }(w)\}$

10.4 Génération pseudo-aléatoire

← 100 chiffres →

Ex: p = premier $\equiv 3 \bmod 4$

q = premier $\equiv 3 \bmod 4$

$0 < s < pq$ et pgcd$(s, pq) = 1$. Alors:

$s \leftarrow (s \times s) \bmod pq$

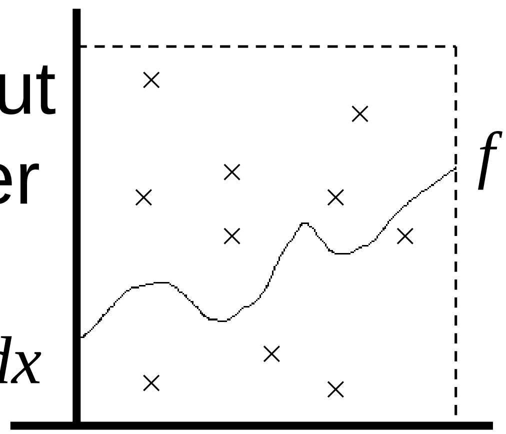
imprimer parité(s)

fournit une suite de bits presque toujours indistinguable d'une suite de bits aléatoire.

10.5 Algos numériques

On veut estimer

$$\int_0^1 f(x)dx$$



$c \leftarrow 0$
pour $i \leftarrow 1$ **jusqu'à** n **faire**
  $x \leftarrow uniforme(0, 1)$
  $y \leftarrow uniforme(0,1)$
  si $y \leq f(x)$ **alors** $c \leftarrow c + 1$
retourner c/n

100 fois plus de points $\Rightarrow$ erreur espérée 10 fois plus petite, c.à.d. un chiffre de précision de plus.

Pour la même précision, un algo déterministe utilisera d points, pour un $d \ll n$.

Mais en s dimensions, pour avoir la même précision avec l'algorithme probabiliste, il faut toujours n points. Par contre, l'algo déterministe aura probablement besoin de d^s points.

$\Rightarrow$ quand s est plus grand que 2 ou 3, l'algo probabiliste l'emporte.

10.6 Algos Monte Carlo

- peut se tromper, mais
- trouve une solution correcte avec bonne probabilité, **quel que soit** l'exemplaire à traiter.

Aucun avertissement en cas d'erreur

p-correct: retourne une solution correcte avec probabilité p ou plus, $0 < p < 1$.

peut être biaisé, i.e. posséder la propriété de ne jamais se tromper sur certaines de ses réponses.

Exemple de Monte Carlo: multiplication de matrices

Pour vérifier si

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 5 & 4 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & 4 & 5 \\ 3 & 2 & 4 \\ 2 & 1 & 3 \end{bmatrix} = \begin{bmatrix} 13 & 11 & 22 \\ 31 & 32 & 58 \\ 23 & 31 & 50 \end{bmatrix}$$

on génère un vecteur X d'éléments 0 et 1, et on vérifie si

$$[\;] \times [\;] \times X = [\;] \times X.$$

Exemple: $X = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 5 & 4 & 3 \end{bmatrix} \begin{bmatrix} 1 & 4 & 5 \\ 3 & 2 & 4 \\ 2 & 1 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \stackrel{?}{=}$$

$$\begin{bmatrix} 13 & 11 & 22 \\ 31 & 32 & 58 \\ 23 & 31 & 50 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 5 & 4 & 3 \end{bmatrix} \begin{bmatrix} 6 \\ 7 \\ 5 \end{bmatrix} \stackrel{?}{=} \begin{bmatrix} 35 \\ 89 \\ 73 \end{bmatrix}$$

L’algorithme retourne **vrai**.

fonction *Freivalds*(A,B,C,n)

vecteur $X[1..n]$

pour $i \leftarrow 1$ **à** n **faire** $X[i] \leftarrow 0$ *ou* 1

si $A \times (B \times X) = C \times X$

alors retourner *vrai*

sinon retourner *faux*

Remarque:

Freivalds(A,B,C,n) *faux* $\Rightarrow$

$A \times B \neq C$ hors de tout doute

mais…

Freivalds(A,B,C,n) *vrai* $\Rightarrow$?

Prob[$A \times B = C$] $\geq$ ½ ?

NON!

La **bonne** façon*:*

$A \times B = C \Rightarrow$
Prob[*Freivalds(*A,B,C,n*) vrai*] *=1*

$A \times B \neq C \Rightarrow$
$$Prob[\underbrace{Freivalds(A,B,C,n)\ faux}_{\text{bonne réponse}}] \geq ½$$

Pourquoi cette dernière inégalité?

<u>Donc</u> *Prob*[bonne réponse] ≥ ½
quel que soit l'exemplaire.

Temps dans $O(n^2)$ mais pas très impressionnant!

Amplification de l'avantage:

fonction *amplif*(A,B,C,n,k)

pour $i \leftarrow 1$ **à** k **faire**

  si *Freivalds*(A,B,C,n) *faux*

  alors retourner *faux*

retourner vrai

$A \times B = C \Rightarrow$

Prob[*amplif*(A,B,C,n,k) *vrai*] $= 1$

$A \times B \neq C \Rightarrow$

Prob[*amplif*(A,B,C,n,k) *faux*] $\geq 1 - 2^{-k}$

Donc: *Prob*[*erreur*] $\leq 2^{-k}$ quel que soit l'exemplaire!

N.B.: $2^{-300} \approx 10^{-100}$.

$$\boxed{\begin{array}{l} A \times B \neq C \Rightarrow \\ \textit{Prob}[\textit{Freivalds}(A,B,C,n) \textit{ faux}] \geq ½? \end{array}}$$

$A \times B \neq C \Rightarrow$
$\exists$ colonne i de $(AB - C)$ non nulle.

Pour chaque choix de vecteur X hormis l'entrée i, on aura:

- $(AB - C)$(« X sans i ») = $\mathbf{0} \Rightarrow$

$$\textit{Prob}[(AB - C)\, X \neq \mathbf{0}\,] = \textit{Prob}[\, X_i = 1]$$
$$= ½$$

- $(AB - C)$(« X sans i ») $\neq \mathbf{0} \Rightarrow$

$$\textit{Prob}[(AB - C)\, X \neq \mathbf{0}\,] \geq \textit{Prob}[X_i = 0]$$
$$= ½$$

Test probabiliste de primalité: essai 1

fonction *premier1(n)*
$d \leftarrow uniforme\,(2\,..\,\lfloor\sqrt{n}\rfloor)$
si ($n \equiv 0 \; mod \; d$) **retourner** *faux*
sinon retourner *vrai*

faux-biaisé: bien!

mais…

retourne presque toujours vrai, premier ou pas, donc peu utile!

Théorème de Fermat

$$p \textit{ premier et } 1 \le a \le p\text{-}1 \quad \Rightarrow \quad a^{p-1} \equiv 1 \bmod p$$

Contraposée:

Soient n *entier et* $1 \le a \le n\text{-}1;$

$\neg(a^{n-1} \equiv 1 \bmod n) \Rightarrow n$ *composé.*

fonction *premier2*(n)
$a \leftarrow uniforme(2\,..\,n-1)$
si ($a^{n-1} \equiv 1$ *mod* n) **retourner** *vrai*
sinon retourner *faux*

Toujours faux-biaisé: bien!
Toujours pas p*-correct car trop de faux témoins de primalité.*

Test de Miller-Rabin

fonction *premier3(n impair > 4)*
$a \leftarrow uniforme\,(2\,..\,n-1)$
retourner *vrai* ***ssi*** $a \in B(n)$

où

$$B(n) = \{\ 2 \le a \le n\ \ :\ \ n-1 = 2^s t \quad \text{et}$$

$$a^t \bmod n = 1$$

$$\text{ou}$$

$$\exists i,\quad 0 \le i < s,\quad a^{2^i t} \bmod n = n-1\ \}$$

Théorème 10.6.2:
- n *premier* $\Rightarrow$ Pr[*vrai*] *=1*
- n *composé* $\Rightarrow$ Pr[*vrai*] *<¼*

$\Rightarrow$ faux-biaisé et ¾-correct!!

Amplifier l'avantage

sans biais: vitesse tortue

Considérons d'abord:

fonction *ridicule*(x)

si *pile* **alors retourner** *vrai*

sinon retourner *faux*

résoud n'importe quoi...tout en étant ½-correct…pourtant on ne gagne rien à répéter cet algo.

$\Rightarrow$ *observation uno: sans biais, impossible d'amplifier à moins d'être* p*-correct pour un* $p > ½$.

Et si $p > ½$*?*

Amplifier l’avantage

sans biais: vitesse tortue

Algo p*-correct,* $p > ½$*, pour un problème de décision: que faire?*

Répéter et prendre la majorité!

Ex: $p = ¾$*, voyons 3 répétitions*

Pr[err, err, err] $= ¼ \times ¼ \times ¼$

Pr[OK, err, err] $= ¾ \times ¼ \times ¼$

Pr[err, OK, err] $= ¼ \times ¾ \times ¼$

Pr[err, err, OK] $= ¼ \times ¼ \times ¾$

Pr[majorité d’erreurs] = $^{10}/_{64} \approx 16\%$

$\Rightarrow$ *avantage passé de* 75% *à* 84%

Amplifier l'avantage

sans biais: vitesse tortue

Général: algo $(\frac{1}{2}+\varepsilon)$*-correct*

Prendre majorité de k *répétitions*

Pr[i OK] $\geq \binom{k}{i}(\frac{1}{2}+\varepsilon)^{i}(\frac{1}{2}-\varepsilon)^{k-i}$

Pr[majorité d'erreurs] $\leq \sum_{i=0}^{k/2}\cdots$

Statistiques: 95% requiert

$$k > 2.706\ (\frac{1}{4\varepsilon^2}-1)$$

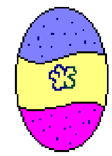
$k=269$ *pour passer de* 55% *à* 95%

Mais 99.5% requiert seulement

$$k > 6.636\ (\frac{1}{4\varepsilon^2}-1)$$

Amplifier l’avantage

avec biais: vitesse lapin



Algo p*-correct, disons faux-biaisé. Que faire?*

Déjà vu! Répéter et répondre vrai à moins d’un seul faux.

Ex: $p = \frac{3}{4}$*, voyons 3 répétitions. Seule possibilité d’erreur est de répondre vrai sur un exemplaire faux, et pour un tel exemplaire:*

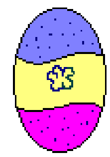
Pr[err, err, err] $= \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4}$

Pr[erreur après 3] = $\frac{1}{64} \approx 2\%$

$\Rightarrow$ *avantage passé de* 75% *à* 98%

Amplifier l'avantage

avec biais: vitesse lapin



Algo p*-correct, faux-biaisé.*

Mieux! Même pas nécessaire que p *soit plus grand que ½.*

Ex: $p = 0.01$

Pr[erreur après k] = $\left(\frac{99}{100}\right)^k$

$\Rightarrow k = 10$ *passe de* 1% *à* $\approx 10\%$

$k = 30$ *passe de* 1% *à* $\approx 25\%$

$k = 150$ *passe de* 1% *à* $\approx 75\%$

Ces valeurs de k *seraient 10 fois moindres en partant de* $p = 0.1$

Algos de Las Vegas

- *Las Vegas de type 1: utilise le hasard pour guider ses choix et arriver infailliblement à résoudre le problème demandé. Mauvais choix $\Rightarrow$ plus long. Ex: tri, hashage.*

- *Las Vegas de type 2: utilise le hasard pour tenter de résoudre un problème, quitte à échouer. Mauvais choix $\Rightarrow$ échec (avoué). Ex: 8 reines, factorisation.*

Las Vegas de type 1

Rappel: sélection et médiane en temps $\Theta(n)$, *en pire cas.*
L'idée était de partitionner sur la base d'une médiane approximée.

Revenons au choix du « pivot »: on peut démontrer que le choix trivial donne un algo ayant

- *temps en* **pire cas** *dans* $\Theta(n^2)$
- *temps en* **moyenne** *dans* $\Theta(n)$.

Un algorithme de Las Vegas de type 1 consiste à choisir le pivot au hasard…et alors?

Sélection et médiane par algo de Las Vegas

- **temps espéré** $\Theta(n)$, précisément égal à ce qu'était *le temps* **en moyenne** *de l'algo à choix trivial*

- *en jouant de malchance, peut prendre autant de temps que le pire cas de l'algo à choix trivial*
- *peut même prendre ce pire temps sur un exemplaire qui aurait été facile pour l'algo à choix trivial!*

- **le grand avantage***? Il n'existe plus de mauvais exemplaire!!!*

Las Vegas de type 1

Particulièrement utile quand un algo déterministe existe, qui est

- *bon en moyenne*
- *mauvais en pire cas*

Alors un algo de Las Vegas pourra

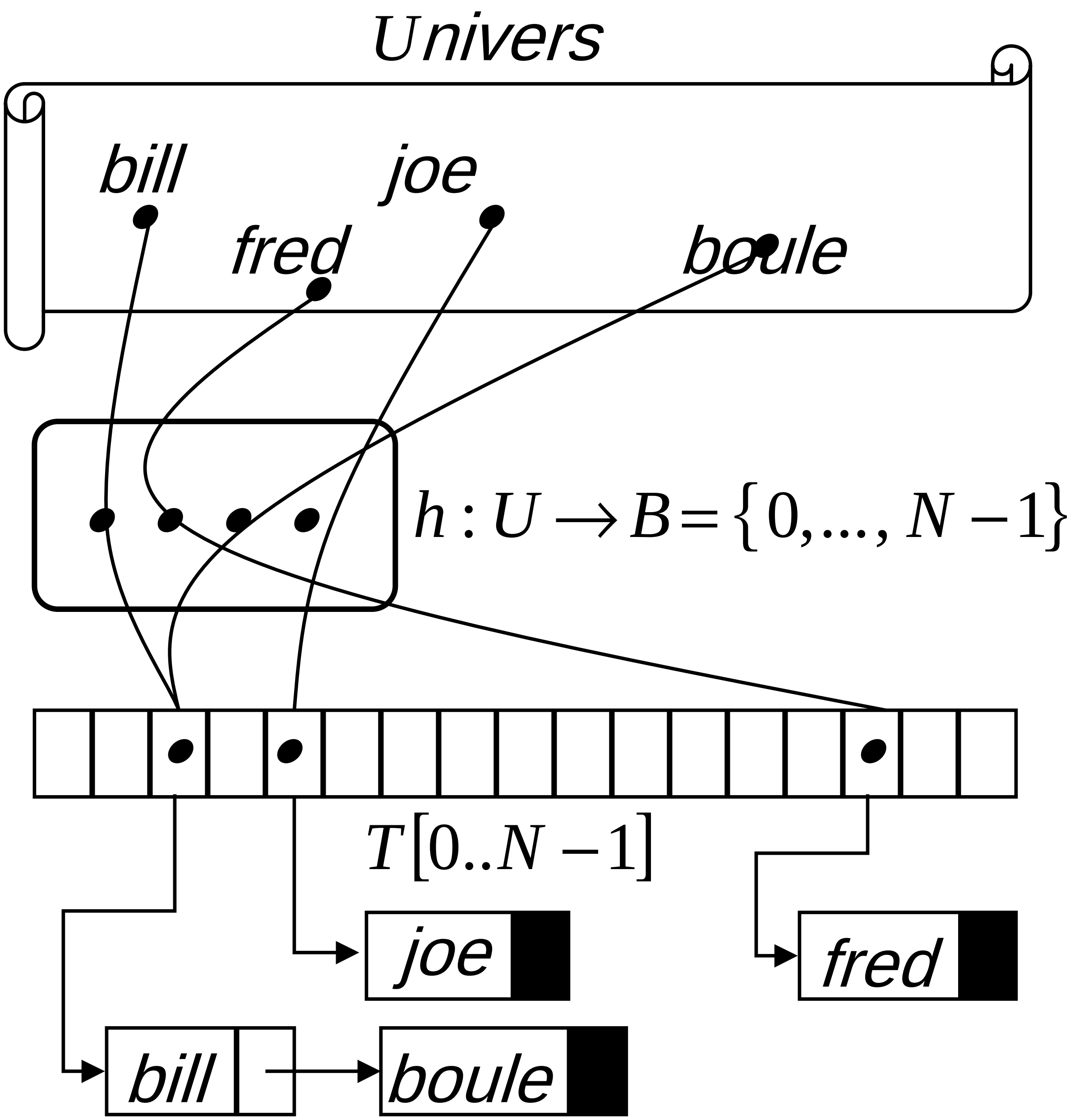
- *éliminer les exemplaires pire cas*
- *uniformiser les exemplaires*
- *maintenir un bon temps espéré*

Autre exemple: quicksort

- $\Theta(n \log n)$ *en moyenne*
- *quadratique en pire cas*
- *devient temps* **espéré** $\Theta(n \log n)$

Las Vegas de type 1

Ex: adressage dispersé universel



m = #{ bill, fred, joe, boule, … }

$\beta = \frac{m}{N}$ = facteur de chargement

Avec fonction de hashage fixée (ex: compilateur),

- *β =longueur moyenne de chacune des N listes à la fin du hashage*
- *temps* **moyen** *requis pour placer les m identificateurs* $\in O(m \times (1+\beta))$
  $\Rightarrow$ linéaire puisque $\beta \leq 1$
- temps **pire cas** $\in \Omega(m^2)$
  (lorsqu'une seule liste à la fin)

Idée Las Vegas de type 1:
Choisir $h : U \to B = \{0,...,N-1\}$
une fonction aléatoire.
Alors
temps **espéré** $\in O(m \times (1+\beta))$
quelle que soit la séquence!

Problème: il y a $\#\{h : U \to B\} = N^{\#U}$
fonctions h *différentes. Le choix aléatoire d'une telle fonction exigerait de spécifier* $(\#U) \times \lg N$
bits. Il en coûterait donc $m \lg N$
pour placer **chaque** *identificateur!*

Que faire?

<u>Définition</u>: l'ensemble de fonctions

$$H \subseteq \{ h : U \rightarrow B \}$$

est une classe <u>universelle2</u> si

$$(\forall x \neq y \in U)$$

$$\Pr_{h \in H} [h(x) = h(y)] \leq 1/N$$

$\Rightarrow$ *quels que soient les deux identificateurs* x *et* y *(distincts), la probabilité de collision est faible.*

<u>Résultat</u>: Quelle que soit la séquence de m *identificateurs, le temps* **espéré** *pour traiter cette séquence* $\in O(m \times (1 + \beta))$.

$\Rightarrow$ *linéaire puisque* β *constant!*

Magie: de telles fonctions existent (et sont calculables efficacement).

Exemple:

$$U = \{0, \ldots, m-1\}$$

$$B = \{0, \ldots, N-1\}$$

$$p \text{ premier} \geq m$$

Alors

$$H = \{ h_{ij} : U \to B \parallel 1 \leq i,\ j < p \}$$

où

$$h_{ij}(x) = ((ix + j) \bmod p) \bmod N$$

est une classe universelle2.

$\Rightarrow$ *facile de tirer aléatoirement une fonction* $h \in H$, *et*

$\Rightarrow$ facile de calculer $h(x)$.

Las Vegas de type 2

Rappel: un tel algo peut échouer, mais détecte alors son échec.

procédure *LV(x,* **var** *y,* **var** *succès)*

Au retour,
succès vrai $\Rightarrow$ y est solution à x
succès faux $\Rightarrow$ pas de chance

$p(x)$ =probabilité de succès, et
$(\forall$ exemplaire $x)[p(x) > 0]$.

Mieux si possible:
$(\exists\delta>0)(\forall$ exemplaire $x)[p(x) > \delta]$.

Répétition non bornée d'un algo de Las Vegas de type 2:

fonction *obstiné*(x)
  répéter
    LV(x, y, *succès*)
  jusqu'à *succès*
  retourner y

Réponse: toujours correcte
toujours obtenue...

…un de ces jours!

…mais quand au juste?

Soient

- p*: probabilité de succès de LV*
- s*: temps espéré de LV en cas de succès*
- e*: temps espéré de LV en cas d'échec*
- t*: temps espéré de obstiné*

Alors

$$t = ps + (1-p)(e+t)$$

d'où

$$t = s + \frac{(1-p)}{p}e$$

<u>*Bref*</u>*:* $s\downarrow,\ e\downarrow,\ p\uparrow \ \Rightarrow\ t\downarrow$

Las Vegas de type 2

Ex: placement des huit reines

Rappel: exploration du graphe des vecteurs k-prometteurs à l'aide d'un algorithme à retour arrière.
k *=8* $\Rightarrow$ *114 explorés parmi 2057.*

Observation: les positions des reines qui résolvent le problème ont l'air plutôt arbitraire.

Approche Las Vegas « vorace »: parmi les positions qui restent, choisir au hasard, et abdiquer si un cul de sac est atteint.

Placement Las Vegas des 8 reines

Avantages:

- *conceptuellement plus simple*
- *plus rapide en principe*

  p*: Pr[succès] =0.1293 (=# de solutions/# total, ordinateur)*

  s*: temps espéré en cas de succès =coût de générer 9 vecteurs*

  e*: temps espéré en cas d'échec =6.971 vecteurs (ordinateur)*

  t*: temps espéré de l'algo*

$$= s + \frac{(1-p)}{p} e = 55.93$$

(comparer aux 114 vecteurs engendrés par retour arrière!)

Placement Las Vegas des 8 reines

En fait: dans le cas de 8 reines, le coût de la génération des nombres aléatoires annule le gain en nombre de vecteurs générés.

Faire mieux?
Oui! En ajustant s, e *et* p.

Idée: générer les k *premières reines aléatoirement, et les* $8-k$ *dernières par retour arrière.*

$k=2$*: trois fois plus rapide que retour arrière*

$k=3$*: seulement 2 fois plus rapide, même si moins de vecteurs*

Placement Las Vegas de n reines

L'avantage de Las Vegas sur le retour arrière croît énormément lorsque n *augmente.*

Ex: $n=39$

- *pur retour arrière:* 10^{10} *vecteurs explorés avant première solution*
- *pur Las Vegas: un million de fois plus rapide en implantation réelle!*
- *hybride avec k=29: 2 millions de fois plus rapide en implantation, et 20 millions moins de vecteurs.*

Ex: $n=1000$

- *bonne idée de choisir* $k=983$ **:-)**

Las Vegas de type 2
Ex: factorisation de grands entiers

- *problème important*
- *aucun algo efficace connu (en fait, la crypto actuelle repose sur sa difficulté!)*
- *ne semble pourtant pas NP-complet*
- *choix aléatoires mêlés à une stratégie judicieuse et à des estimés sophistiqués tirés de la théorie des nombres permettent parfois de réussir.*