L'aléa du calcul et la vérification interactive

Adam Shimi

2 avril 2019

Plan

1. Les calculs probabilistes
2. Calculer efficacement avec l'aléa
3. Protocoles interactifs : la valeur de l'aléa pour les preuves

Qu'est-ce que le calcul probabiliste nous apporte ?

Intuition

- Un algorithme avec la possibilité de faire des choix probabilistes. Comme s'il lançait des pièces.
- Un compromis entre déterminisme et non-déterminisme : il ne faut pas forcément toujours accepter, mais il faut accepter "suffisament souvent".
- Moyen de ne pas surdéterminer un algorithme, en faisant des choix aléatoires plutôt que déterministes.

Deux formulations équivalentes de l'aléa

En ligne et hors ligne

- En ligne : le lancer de pièces. Lorsqu'il y a un choix aléatoire à faire, on "lance des pièces". C'est-à-dire qu'on tire un bit aléatoire.
  Comme une machine non-déterministe aux choix probabilistes.
- Hors ligne : une liste initiale de bits aléatoires. Les choix sont fait en lisant un nouveau bit de cette liste.
  Comme une machine déterministe qui vérifie un certificat généré aléatoirement.

Equivalence des deux formulations

La même intuition que pour $\mathcal{NP}$ nous permet de passer d'une formulation à l'autre : la liste initiale peut encoder tous les lancers de pièces, et les lancers de pièces peuvent simuler une lecture dans la liste initiale.

Las-vegas et Monte-Carlo

Les deux variations d'algorithmes probabilistes

- Algorithmes de Las Vegas : des algorithmes qui renvoient toujours la réponse correcte, mais dont le temps pris pour le calcul est une variable aléatoire.
- Algorithmes de Monte Carlo : des algorithmes qui finissent en temps déterministe (en fonction de la taille), mais qui peuvent retourner des erreurs.

Puisque l'on étudie des classes de complexité, on veut des garanties sur le temps pris par l'algorithme et on se limitera donc aux algorithmes de Monte Carlo.

Types d'erreurs

Double erreur et simple erreur

Une des charactéristiques fondamentales des calculs probabilistes est la possibilité d'erreur. Pour autant, il y a différents types d'erreurs que l'on peut considérer.

- Double erreur : Il y peut y avoir une erreur en acceptant et en rejettant. Donc l'algorithme peut accepter une mauvaise instance, et en rejetter une bonne.
- Simple erreur : Il ne peut y avoir une erreur qu'en acceptant ou en rejettant, pas pour les deux. Cela donne deux variations, les algorithmes qui ne peuvent se tromper qu'avec des faux négatifs et ceux avec des faux positifs.

Digression : complexité lissée (smoothed complexity)

De par notre définition classique de la complexité, il est possible qu'un algorithme soit efficace partout sauf pour certaines instances très spécifiques.
Faut-il pour autant dire qu'il n'est pas efficace ?

Utiliser l'aléa pour négliger les cas pathologiques

La complexité lisée essaye de résoudre ce problème : le temps mesuré pour une entrée devient l'espérance du temps quand l'entrée est perturbée aléatoirement (par une distribution gaussienne).
On prend ensuite le maximum de ces temps pour chaque taille d'entrée.

Si les entrées qui prennent du temps sont "pathologiques" (très spécifiques), alors la pertubation fera baisser énormément la complexité.

Plan

1 Les calculs probabilistes

2 Calculer efficacement avec l'aléa

3 Protocoles interactifs : la valeur de l'aléa pour les preuves

La classe $\mathcal{BPP}$

Définition

Un language $L \subseteq \{0,1\}^*$ appartient à la classe de complexité $\mathcal{BPP} \triangleq \exists M$ une machine de Turing polynomiale, $\exists p$ un polynome :

- $\forall x \in L : |\{y \in \{0,1\}^{p(|x|)} | M(\langle x, y \rangle)\}| \geq 2/3$.
- $\forall x \notin L : |\{y \in \{0,1\}^{p(|x|)} | M(\langle x, y \rangle)\}| \leq 1/3$.

En donnant la même probabilité $\frac{1}{p(|x|)}$ à tous les y, on obtient une machine acceptant x avec probabilité $2/3$ s'il est dans L, et le rejettant avec probabilité $2/3$ s'il n'y est pas.

La non-importance de la constante

Pourquoi utiliser $2/3$ dans la définition de $\mathcal{BPP}$?
Juste une convention : toute constante $> 1/2$ définit la même classe.

Réduction de l'erreur

Comment réduire l'erreur

Pour réduire l'erreur, il suffit de recommencer l'algorithme plusieurs fois et de décider la réponse majoritaire.

- Pour $k > 0$, on définit $X_1, ..., X_k$ des variables aléatoires qui valent 1 si l'algorithme donne la bonne réponse et 0 sinon.
- Pour une probabilité d'erreur $p < 1/3$, l'espérance des X_i est p et l'espérance de $X = \sum_{1 \leq i \leq k} X_i$ est $\mu = k * p \geq (2/3)k$.
- $Pr[X < k/2] \leq Pr[X < (3/4)\mu] = Pr[|X - \mu| > (1/4)\mu]$.
- Par un résultat de théorie des probabilités, cette quantité est majorée par $2e^{-(1/4^2)*(1/3)*\mu} \leq 2e^{-(1/4^2)*(1/3)*(2/3)*k}$.

La probabilité d'erreur décroit donc comme une exponentielle négative de k. Il faut donc très peu de répétitions pour obtenir une probabilité d'erreur négligeable.

Test de primalité (1/2)

Le problème

Le test de primalité, comme son nom l'indique, est le problème de décision contenant tous les nombres premiers.
Rappel : Un nombre entier est premier s'il n'est divisible que par 1 et par lui-même.

Résultats de théorie des nombres (admis)

- Pour $p > 2$ premier, $\forall r \in [1, ..., p-1]$, l'équation $x^2 \equiv r^2 (mod\ p)$ a exactement deux solutions r et $-r \equiv p - r (mod\ p)$.
- Pour N un entier composite, impair et différent d'une puissance entière, $\forall r \in [1, ..., p-1]$, l'équation $x^2 \equiv r^2 (mod\ p)$ a au moins quatres solutions modulo p.

Test de primalité (2/2)

L'algorithme est le suivant :

- Vérifier si N est pair ou s'il est une puissance entière. Si c'est le cas, le rejeter.
- Choisir uniformément $r \in [1, ..., N-1]$ et poser $s \equiv r^2 (mod\ p)$.
- Calculer probabilistiquement une racine carrée r' de s modulo p. Si $r' \equiv \pm r (mod\ p)$, accepter ; sinon, rejeter.

Analyse

Si le nombre est premier, il ne peut pas y avoir d'erreur. Et s'il ne l'est pas, alors il y a une probabilité au plus $1/2$ d'obtenir $\pm r$. Enfin, la vérification initiale et le calcul de la racine carrée se font probabilistiquement en temps polynomial.

Les classes $\mathcal{RP}$ et $co\mathcal{RP}$

Définition de $\mathcal{RP}$

Un language $L \subseteq \{0,1\}^*$ appartient à la classe de complexité $\mathcal{RP} \triangleq \exists M$ une machine de Turing polynomiale, $\exists p$ un polynome :

- $\forall x \in L : |\{y \in \{0,1\}^{p(|x|)} | M(\langle x, y \rangle)\}| \geq 1/2.$
- $\forall x \notin L : |\{y \in \{0,1\}^{p(|x|)} | M(\langle x, y \rangle)\}| = 0.$

Définition de $co\mathcal{RP}$

Un language $L \subseteq \{0,1\}^*$ appartient à la classe de complexité $co\mathcal{RP} \triangleq \exists M$ une machine de Turing polynomiale, $\exists p$ un polynome :

- $\forall x \in L : |\{y \in \{0,1\}^{p(|x|)} | M(\langle x, y \rangle)\}| = 1.$
- $\forall x \notin L : |\{y \in \{0,1\}^{p(|x|)} | M(\langle x, y \rangle)\}| \leq 1/2.$

$co\mathcal{RP}$ est évidemment le complémentaire de $\mathcal{RP}$.

Réduction de l'erreur pour $\mathcal{RP}$

A l'instar de $\mathcal{BPP}$, la constante importe peu. Ici, il faut juste une probabilité d'erreur < 1.

Comment réduire l'erreur

Pour réduire l'erreur, il suffit de recommencer l'algorithme plusieurs fois. Si au moins une exécution accepte, on accepte. Sinon, on rejette

- Si $x \notin L$, alors aucune exécution n'acceptera et on rejettera nécessairement.
- Si $x \in L$, il y a une probabilité $\leq 2^{-k}$ que chacune des k exécutions ait rejeté par erreur.

La probabilité d'erreur décroit ici aussi comme une exponentielle négative de k. Il faut donc très peu de répétitions pour obtenir une probabilité d'erreur négligeable.

Test d'égalité de polynomes : PIT

Le problème

Soit F un corps fini (par exemple les entiers modulo 5). Soit $p, q : F^n \mapsto F$ deux polynomes de degré au plus d. Le test d'égalité demande si $\forall x_1, ..., x_n \in F : p(x_1, ..., x_n) = q(x_1, ..., x_n)$.

Intuition

Demander si deux polynomes sont égaux revient à demander si leur différence est nulle.
Il suffit donc de tester la valeur en un point choisi aléatoirement de $p(n) - q(n)$.

- Si le résultat est non nul, on est sûr de devoir rejeter.
- Sinon, il y a une probabilité $\frac{d}{|F|}$ d'être tombé sur une racine du polynome, puisqu'il y en a au plus d.

Liens entres les classes de complexité probabilistes

Quelques inclusions évidentes

- $\mathcal{RP} \subseteq \mathcal{BPP}$
- $co\mathcal{RP} \subseteq \mathcal{BPP}$

Conjectures

- $\mathcal{P} = \mathcal{BPP}$
- $\mathcal{RP} = \mathcal{BPP}$
- $co\mathcal{RP} = \mathcal{BPP}$

Elles dérivent toutes de la première, puisque si $\mathcal{P} = \mathcal{BPP}$, alors il existe une version déterministe et polynomiale de chaque algorithme probabiliste polynomial.

Dérandomization

Du probabiliste au déterministe

Dérandomization : Partir d'un algorithme probabiliste polynomial pour en obtenir un déterministe qui soit toujours polynomial.

Des générateurs pseudo-aléatoires à la dérandomization

Imaginons l'existence d'une machine de Turing polynomiale déterministe telle que sa sortie soit indifférentiable, en temps polynomial, de bits aléatoires.
On pourrait alors utiliser ce générateur pour créer déterministiquement les lancers de pièces d'un algorithme probabiliste, et donc dérandomizer tous les algorithmes probabilistes.

On conjecture que la dérandomization est toujours possible parce que l'on conjecture l'existence de tels générateurs pseudo-aléatoires.

Plan

1 Les calculs probabilistes

2 Calculer efficacement avec l'aléa

3 Protocoles interactifs : la valeur de l'aléa pour les preuves

De l'utilité de poser des questions

À quoi servent les enseignants ?

- À quelque chose ? (on espère ...)
- À motiver les étudiants ? (peut-être)
- À répondre aux questions ?

Reformulons : quelle est le pouvoir des questions pour la vérification ?

Formalisons une interaction

Interactions de fonctions déterministes

Soit $f, g : \{0,1\}^* \mapsto \{0,1\}^*$ deux fonctions, et k un entier (qui peut dépendre de x).
Alors une interaction en k rounds de f et g sur $x \in \{0,1\}^* \triangleq$ les $a_1, ..., a_k \in \{0,1\}^*$ telles que :

- $a_1 = f(x)$
- $a_2 = g(x, a_1)$
- $a_{2i+1} = f(x, a_1, ..., a_{2i})$
- $a_{2i+2} = g(x, a_1, ..., a_{2i+1})$

La sortie de f à la fin de l'interaction est
$out_f\langle g, x\rangle(x) \triangleq f(x, a_1, ..., a_k)$. Et symétriquement pour g.

a_1 représente le premier message que f envoie, a_2 la réponse de g après avoir reçu a_1, et ainsi de suite.

Les protocoles interactifs déterministes : $d\mathcal{IP}$

La classe $d\mathcal{IP}$

Un language $L \subseteq \{0,1\}^*$ appartient à la classe de complexité $d\mathcal{IP} \triangleq \exists V$ une machine de Turing polynomiale sur les entrées $x, a_1, ..., a_i$ qui peut avoir une interaction polynomiale avec n'importe quelle fonction P telle que

- $x \in L \implies \exists P : \{0,1\}^* \mapsto \{0,1\}^* out_V\langle V, P\rangle(x) = 1.$
- $x \in L \implies \forall P : \{0,1\}^* \mapsto \{0,1\}^* out_V\langle V, P\rangle(x) = 0.$

Le vérifieur V accepte x au terme d'une interaction polynomiale avec un prouveur P ssi $x \in L$

Important : nous n'imposons aucune contrainte sur P.

Les interactions déterministes n'ajoutent rien aux preuves

$d\mathcal{IP} = \mathcal{NP}$

- Il est évident que tout langage dans $\mathcal{NP}$ peut être vérifié en 1 round, avec V son vérifieur et P une fonction donnant le bon certificat si $x \in L$.
- A l'inverse, avec les $a_1, ..., a_k$ faisant accepter le vérifieur lors de l'interaction, on peut construire un vérifieur polynomial.

Parce qu'ils sont de taille polynomiale, les messages de l'interaction représentent un certificat, et inversement.

Et si l'on ne cherchait pas l'exactitude parfaite ?

Interactions probabiliste de fonctions

Soit $f, g : \{0,1\}^* \mapsto \{0,1\}^*$ deux fonctions, k et m deux entiers et r une variable aléatoire dans $\{0,1\}^m$.
Alors une interaction probabiliste en k rounds de f et g sur $x \in \{0,1\}^* \triangleq$ les variables aléatoires $a_1, ..., a_k$ telles que :

- $a_1 = f(x, r)$
- $a_2 = g(x, a_1)$
- $a_{2i+1} = f(x, r, a_1, ..., a_{2i})$
- $a_{2i+2} = g(x, a_1, ..., a_{2i+1})$

La sortie de f à la fin de l'interaction est la variable aléatoire $out_f\langle g, x\rangle(x)$ dépendant de r.

La seule différence vient des bits aléatoires dans r, qui permettent à f de répondre probabilistiquement.

Les protocoles probabilistes : $\mathcal{IP}$

La classe $\mathcal{IP}$

Un language $L \subseteq \{0,1\}^*$ appartient à la classe de complexité $\mathcal{IP} \triangleq \exists V$ une machine de Turing polynomiale sur les entrées $x, r, a_1, ..., a_i$ qui peut avoir une interaction polynomiale avec n'importe quelle fonction P telle que

- $x \in L \implies \exists P : \{0,1\}^* \mapsto \{0,1\}^*$
  $Pr[out_V\langle V, P\rangle(x) = 1] \geq 2/3$
- $x \notin L \implies \forall P : \{0,1\}^* \mapsto \{0,1\}^*$
  $Pr[out_V\langle V, P\rangle(x) = 1] \leq 1/3$

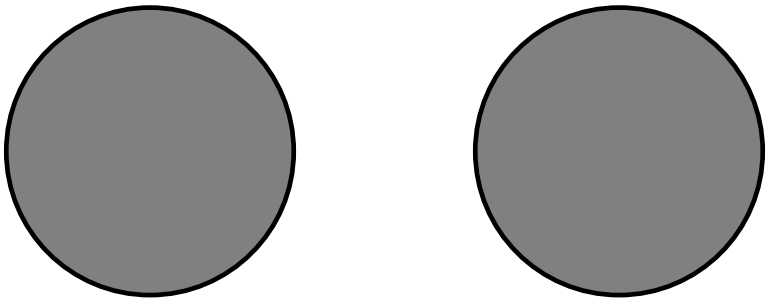
Les bits aléatoires sont utilisés par le vérifieur pour poser des questions moins prévisibles (car pas déterministes).
Pour autant, cela permet-il de vérifier plus rapidement ?

Exemple : le protocole du daltonien

Convaincre le daltonien

V a deux balles "identiques", mais il est daltonien. P lui assure qu'elles ont des couleurs différentes. Comment V peut-il le vérifier ?
Il suffit de lancer à répétition des pièces (en secret), d'inverser les deux balles si pile et de les laisser si face.
P doit ensuite dire si les balles ont été inversées ou non.

- S'il se trompe, V rejette avec certitude.
- Sinon, V accepte, puisque sur n lancers la probabilité que P ne fasse pas d'erreur avec des balles identiques est $\frac{1}{2^n}$.

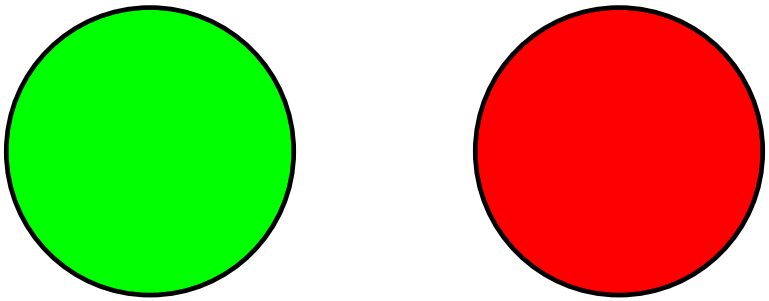


Exemple : le protocole du daltonien

Convaincre le daltonien

V a deux balles "identiques", mais il est daltonien. P lui assure qu'elles ont des couleurs différentes. Comment V peut-il le vérifier ?
Il suffit de lancer à répétition des pièces (en secret), d'inverser les deux balles si pile et de les laisser si face.
P doit ensuite dire si les balles ont été inversées ou non.

- S'il se trompe, V rejette avec certitude.
- Sinon, V accepte, puisque sur n lancers la probabilité que P ne fasse pas d'erreur avec des balles identiques est $\frac{1}{2^n}$.

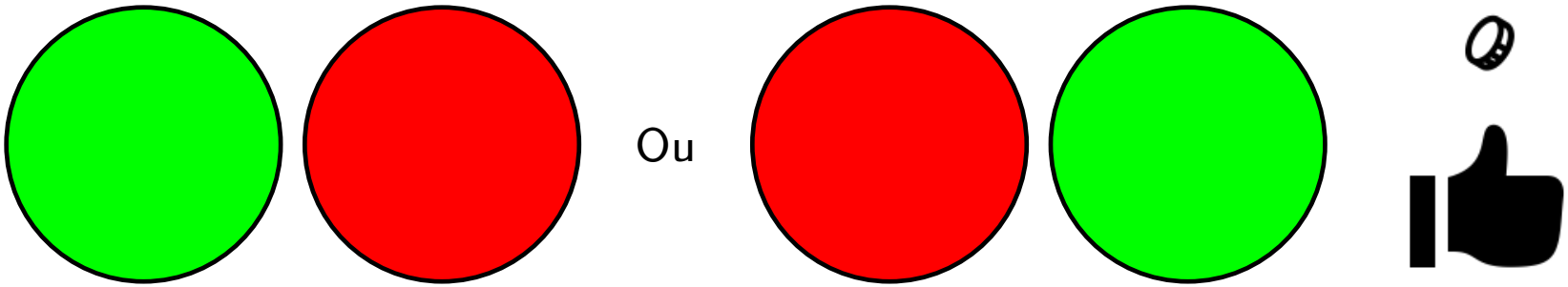


Exemple : le protocole du daltonien

Convaincre le daltonien

V a deux balles "identiques", mais il est daltonien. P lui assure qu'elles ont des couleurs différentes. Comment V peut-il le vérifier ?
Il suffit de lancer à répétition des pièces (en secret), d'inverser les deux balles si pile et de les laisser si face.
P doit ensuite dire si les balles ont été inversées ou non.

- S'il se trompe, V rejette avec certitude.
- Sinon, V accepte, puisque sur n lancers la probabilité que P ne fasse pas d'erreur avec des balles identiques est $\frac{1}{2^n}$.



Application : le non-isomorphisme de graphes

Le problème

Le non-isomorphisme de graphe est le problème de décision défini par les paires de graphes $\langle G, H \rangle$ telles que G n'est pas isomorphe à H.
Rappel : deux graphes sont isomorphes s'il est possible de transformer l'un en l'autre en renommant les sommets.

Le protocole des daltoniens vérifie le non-isomorphisme

Si l'on remplace les balles par les graphes, un prouveur sans limite de calcul saura toujours différencier deux graphes non-isomorphes.
Sinon, il a au mieux une chance sur 2 de deviner l'inversion.
Le non-isomorphisme de graphe appartient donc à $\mathcal{IP}$.

Le pouvoir de l'interaction et de l'aléa

Est ce que $\mathcal{IP} \supsetneq \mathcal{NP}$? De manière équivalente, est-ce que l'aléa donne du pouvoir à la vérification interactive ?

$\mathcal{IP} = \mathcal{PSPACE}$

Un résultat récent étonnant est que $\mathcal{IP} = \mathcal{PSPACE}$. Etonnant parce que les conjectures actuelles sont que $\mathcal{PSPACE}$ est bien plus grand que $\mathcal{NP}$.
La combinaison de l'interaction et de l'aléa donne donc une puissance supplémentaire à la vérification.

Ici encore, la preuve est trop compliquée pour être traitée durant ce cours, mais elle est disponible dans les cours de référence.
Ce résultat signifie notamment qu'un prouveur "tout puissant" (comme un alien ou Dieu) pourrait nous convaincre en un nombre polynomial de questions qu'il a une stratégie gagnante aux échecs ou au go.

Prouver sans donner d'information : les protocoles zero-knowledge

On peut considérer le vérifieur comme un attaquant essayant d'obtenir des informations en vérifiant par exemple une clé.

Du point de vue de la sécurité

La solution est d'utiliser un protocole zero-knowledge, un protocole interactif tel que : pour tout vérifieur polynomial V', il existe un algorithme probabiliste polynomial S tel que $out_{V'}\langle P, V'\rangle$ suit la même distribution que $S(x)$.

Quelque soit la stratégie du vérifieur, il n'obtient donc aucune information sur le prouveur, puisqu'il pourrait juste faire tourner l'algorithme S à la place de l'interaction.
Le protocole pour le non-isomorphisme de graphe est un exemple parfait, puisque le vérifieur demande une information qu'il connaît déjà (si les graphes ont été inversés ou non).

Lancers de pièces publics : $\mathcal{AM}$

Que se passe-t-il si les bits aléatoires du vérifieur sont connus du prouveur ?

Arthur et Merlin : la classe $\mathcal{AM}$

Un language L appartient à la classe $\mathcal{AM}[k] \triangleq$ il existe un protocol interactif probabiliste vérifiant L en k rounds, avec les contraintes suivantes :

- Le vérifieur n'envoie que des bits aléatoires
- Le vérifieur ne peut utiliser que les bits aléatoires qu'il a envoyé pour ses décision probabilistes.

Le nom de la classe vient d'Arthur, légendaire roi de Bretagne, et de Merlin, son magicien tout-puissant. Ici Arthur est le vérifieur, et il est complètement incapable de cacher ses lancers de pièce à la magie de Merlin.

Simuler des lancers de pièces privés

Quelques résultats sur $\mathcal{AM}$

- $\forall k \in \mathbb{N} : \mathcal{IP}[k] \subseteq \mathcal{AM}[k+2]$
- $\mathcal{AM}[poly(n)] = \mathcal{IP}$
- $\forall k \in \mathbb{N} : \mathcal{AM}[k] = \mathcal{AM}[2]$

Intuition pour le non-isomorphisme de graphes dans $\mathcal{AM}[2]$:

- Imaginons que le vérifieur avec pièces privées envoie comme message un des graphes renommés, selon le résultat du lancer.
- Si les graphes sont non-isomorphes, alors il y a à peu près $2n!$ graphes qui sont envoyables. Sinon, il n'y en a que $n!$.
- Le prouveur avec des pièces publiques va certifier que l'ensemble des messages que le verifieur aurait pu envoyer (les graphes permutés) est de taille $> n!$.

Dans l'idée, le prouveur avec pièces publiques certifie que le vérifieur avec pièces privées aurait accepté.

Conclusion

Résumé

- Les algorithmes probabilistes étudiées en théorie de la complexité rajoutent des bits aléatoires à l'entrée.
- On définit plusieurs classes efficaces et probabilistes selon le genre d'erreur toléré : $\mathcal{BPP}, \mathcal{RP}$ et $co\mathcal{RP}$.
- La conjecture est que $\mathcal{P} = \mathcal{BPP}$, ce qui découle de la conjecture d'existence de générateurs pseudo-aléatoires.
- Les protocoles interactifs probabilistes sont plus puissant que les protocoles déterministes : $d\mathcal{IP} = \mathcal{NP}$ et $\mathcal{IP} = \mathcal{PSPACE}$.
- Les protocoles interactifs probabilistes sont aussi puissants avec des pièces publiques qu'avec des pièces privées : $\mathcal{AM}[poly(n)] = \mathcal{IP}$.