Le gros lot

Par Enigmath, le 7 janvier 2026

Énoncé

Vous participez à une loterie un peu particulière. Il y a n tickets à gratter, numérotés de 1 à n. Parmi eux, un seul ticket cache le *gros lot*, tandis que tous les autres ne donnent rien.
Avant même d'acheter un ticket i, vous pouvez consulter deux informations inscrites au dos du ticket :

— le prix d'achat c_i du ticket,
— la probabilité p_i que ce ticket contienne le gros lot.

Vous pouvez acheter et gratter des tickets *dans l'ordre de votre choix*.

Questions :

1. 🌶[1.5] Dans quel ordre devez-vous acheter et gratter les n tickets afin de minimiser le coût espéré pour remporter le gros lot ?
2. 🌶[2.4] Sachant que la même loterie est organisée **tous les mois**, quelle stratégie devez-vous adopter pour minimiser le coût espéré avant de gagner le gros lot au moins une fois ?
3. 🌶[1.9] Si le gros lot vaut g, quelle stratégie faut-il adopter pour maximiser le gain total espéré ?

Solution

1. Pour chaque ticket i, définissons le ratio $r_i = \frac{p_i}{c_i}$. Il faut alors acheter et gratter les tickets dans l'ordre décroissant des r_i. En effet, supposons sans perte de généralité que $r_1 \geq r_2 \geq \cdots \geq r_n$. Si σ est une permutation des tickets correspondant à l'ordre dans lequel vous les grattez, alors le coût espéré avant de gagner vaut $C(\sigma)$, avec C la fonction définie sur l'ensemble des arrangements[1] de $\{1, \ldots, n\}$ par :

$$C(s_1, \ldots, s_k) \triangleq \sum_{i=1}^{k} p_{s_i} \sum_{j=1}^{i} c_{s_j}.$$

On peut montrer que

$$C(\sigma) - C(\mathrm{id}_n) = \sum_{\{i,j\} \in I(\sigma),\, i<j} c_i c_j (r_i - r_j) \geq 0,$$

où $I(\sigma)$ est l'ensemble des **inversions** de σ, c'est-à-dire des paires $\{i, j\}$ telles que $i < j$ et $\sigma(i) > \sigma(j)$. Ainsi, toute permutation σ différente de l'identité ne peut qu'augmenter le coût espéré. L'ordre décroissant des r_i est donc **optimal**.

1. On aurait pu définir C uniquement sur l'ensemble des permutations, mais le cadre plus général des arrangements sera utile pour les questions suivantes.

2. On peut supposer sans perte de généralité que les probabilités p_i sont toutes strictement positives. En effet, si un ticket i vérifie $p_i = 0$, alors il ne peut jamais contenir le gros lot et peut donc être exclu du problème sans modifier la solution optimale. À chaque mois t, on note S_t l'arrangement des tickets qu'on prévoit de gratter (si le gros lot ne se trouve pas dans S_1, alors aucun ticket supplémentaire n'est acheté au cours de ce mois et on attend le mois suivant pour recommencer la procédure en itérant sur les tickets de S_2, et ainsi de suite jusqu'à ce que le gros lot soit remporté). L'objectif est de choisir une stratégie $(S_1, S_2, \dots)$ minimisant le coût total espéré

$$\begin{aligned}\mathcal{J}(S_1, S_2, \dots) &\triangleq \sum_{t=1}^{\infty} \left(\sum_{j \in S_t} p_j \sum_{u=1}^{t-1} \sum_{j \in S_u} c_j + C(S_t) \right) \prod_{u=1}^{t-1} \left(1 - \sum_{j \in S_u} p_j \right) \\ &= C(S_1) + \left(1 - \sum_{j \in S_1} p_j \right) \left(\sum_{j \in S_1} c_j + \mathcal{J}(S_2, \dots) \right),\end{aligned}$$

où C est définie à la question précédente. En notant $\mathcal{J}^* \triangleq \mathcal{J}(S_1^*, S_2^*, \dots)$ la valeur minimale de la fonction objectif $\mathcal{J}$, la relation précédente implique que

$$\mathcal{J}^* = \frac{C(S_1^*) + \left(1 - \sum_{j \in S_1^*} p_j\right) \sum_{j \in S_1^*} c_j}{\sum_{j \in S_1^*} p_j} = \mathcal{J}(S_1^*, S_1^*, \dots).$$

Cela montre qu'il existe une stratégie optimale qui est **stationnaire**, c'est-à-dire de la forme $(S^*, S^*, \dots)$. On note désormais

$$J(S) \triangleq \mathcal{J}(S, S, \dots) = \frac{C(S) + \left(1 - \sum_{j \in S} p_j\right) \sum_{j \in S} c_j}{\sum_{j \in S} p_j} = \frac{\sum_{i=1}^{k} c_{s_i} \left(1 - \sum_{j=1}^{i-1} p_{s_j}\right)}{\sum_{i=1}^{k} p_{s_i}},$$

pour tout arrangement $S = (s_1, \dots, s_k)$ des tickets (avec comme convention $J(\emptyset) = \infty$).

Lemme. *Soit $S = (s_1, \dots, s_k)$, pour $k \geq 1$, et $s_{k+1} \notin S$. Si $r_{s_{k+1}} \geq r_{s_k}$, alors*

$$J(S) \geq \min\left(J(s_1, \dots, s_{k-1}), J(s_1, \dots, s_k, s_{k+1})\right).$$

Démonstration. Si $k = 1$, alors, en utilisant l'hypothèse $r_{s_{k+1}} \geq r_{s_k}$, on a

$$J(s_k, s_{k+1}) - J(s_k) = \frac{p_{s_{k+1}}}{p_{s_k} + p_{s_{k+1}}} \left(\frac{1 - p_{s_k}}{r_{s_{k+1}}} - \frac{1}{r_{s_k}} \right) \leq 0.$$

On suppose donc que $k \geq 2$. Si $J(s_1, \dots, s_{k-1}) < J(S)$, il n'y a rien a montrer. Sinon,

$$J(S) \geq \frac{1}{p_{s_k}} \left(J(S) \sum_{i=1}^{k} p_{s_i} - J(s_1, \dots, s_{k-1}) \sum_{i=1}^{k-1} p_{s_i} \right) = \frac{1 - \sum_{j=1}^{k-1} p_{s_j}}{r_{s_k}} \geq \frac{1 - \sum_{j=1}^{k} p_{s_j}}{r_{s_k}}.$$

On a donc

$$\begin{aligned}J(s_1, \dots, s_k, s_{k+1}) - J(S) &= \frac{J(S) \sum_{i=1}^{k} p_{s_i}}{\sum_{i=1}^{k+1} p_{s_i}} + \frac{p_{s_{k+1}} \left(1 - \sum_{i=1}^{k} p_{s_i}\right)}{r_{s_{k+1}} \sum_{i=1}^{k+1} p_{s_i}} - J(S) \\ &= \frac{-J(S) p_{s_{k+1}}}{\sum_{i=1}^{k+1} p_{s_i}} + \frac{p_{s_{k+1}} \left(1 - \sum_{i=1}^{k} p_{s_i}\right)}{r_{s_{k+1}} \sum_{i=1}^{k+1} p_{s_i}} \\ &= \frac{p_{s_{k+1}}}{\sum_{i=1}^{k+1} p_{s_i}} \left(\frac{1 - \sum_{i=1}^{k} p_{s_i}}{r_{s_{k+1}}} - J(S) \right) \\ &\leq \frac{p_{s_{k+1}}}{\sum_{i=1}^{k+1} p_{s_i}} \left(\frac{1 - \sum_{i=1}^{k} p_{s_i}}{r_{s_{k+1}}} - \frac{1 - \sum_{j=1}^{k} p_{s_j}}{r_{s_k}} \right) \leq 0,\end{aligned}$$

en utilisant l'hypothèse $r_{s_{k+1}} \geq r_{s_k}$. □

Théorème. *Supposons de nouveau sans perte de généralité que $r_1 \geq r_2 \geq \cdots \geq r_n$. Alors, il existe un minimiseur de J de la forme $S^* = (1, 2, \ldots, k)$ pour un certain $k \in \{1, \ldots, n\}$.*

Démonstration. À support fixé $S = (s_1, \ldots, s_k)$, et en tant que fonction de l'ordre, J dépend linéairement de C. D'après la question précédente, un ordre optimal est donné par $s_{i_1} < s_{i_2} < \cdots < s_{i_k}$, c'est-à-dire l'ordre décroissant des ratios r_i. On peut donc se restreindre à l'ensemble $\mathcal{S}^*$ des minimiseurs de J respectant cet ordre. Pour $\ell \geq 1$, on note $\mathcal{S}^*_\ell$ l'ensemble des arrangements $S \in \mathcal{S}^*$ commençant par $(1, 2, \ldots, \ell - 1)$. Soit ℓ le plus grand entier tel que $\mathcal{S}^*_\ell \neq \varnothing$, et soit $S^* \in \mathcal{S}^*_\ell$ de cardinal minimal. Supposons par l'absurde que

$$n > \ell, \qquad S^* = (1, \ldots, \ell - 1, s_\ell, \ldots, s_{|S^*|}), \qquad |S^*| \geq \ell.$$

D'après le lemme précédent, et puisque $r_\ell \geq r_{s_{|S^*|}}$, on a

$$J(S^*) \geq \min\big(J(1, \ldots, \ell - 1, s_\ell, \ldots, s_{|S^*|-1}), J(1, \ldots, \ell - 1, s_\ell, \ldots, s_{|S^*|}, \ell)\big)\,,$$

et donc l'égalité est atteinte. L'arrangement $(1, \ldots, \ell - 1, s_\ell, \ldots, s_{|S^*|}, \ell)$ ne respect pas l'ordre des ratios, mais on peut y remédier en déplaçant ℓ immédiatement après $\ell - 1$, ce qui ne peut que diminuer la valeur de J d'après la question précédente. On obtient ainsi

$$J(S^*) = \min\big(J(1, \ldots, \ell - 1, s_\ell, \ldots, s_{|S^*|-1}), J(1, \ldots, \ell, s_\ell, \ldots, s_{|S^*|})\big)\,.$$

Par conséquent, l'un des deux arrangements du membre de droite appartient à $\mathcal{S}^*$. Or, le premier appartient à $\mathcal{S}^*_\ell$ et contredit la minimalité du cardinal de S^*, tandis que le second appartient à $\mathcal{S}^*_{\ell+1}$, contredisant la maximalité de ℓ. Ceci conclut la contradiction. □

Remarque. *Il découle de ce qui précède que, pour déterminer une stratégie optimale, il suffit d'évaluer J sur les n arrangements de la forme $(1, \ldots, k)$, $k \in \{1, \ldots, n\}$. Autrement dit, parmi toutes les stratégies possibles, une stratégie optimale est nécessairement obtenue en considérant un préfixe de l'ordre optimal.*

Le code Python ci-dessous présente une implémentation de l'algorithme résultant de cette analyse théorique.

python — Télécharger le code

```python
import numpy as np

def J(S,p,c):
    return sum(c[S[i]]*(1 - sum(p[S[:i]])) for i in range(len(S)))/sum(p[S])

def optimal_strategy(p, c, n):
    order = np.argsort(-p/c)
    J_values = []
    for k in range(n):
        S = order[:k+1]
        J_values.append((J(S,p,c),S))
    J_opt, S_opt = min(J_values)
    return J_opt, S_opt

n = 100

p = np.array([
    0.00710623, 0.00413283, 0.00849038, 0.00822335, 0.01067269,
    0.01006182, 0.00144579, 0.0017939 , 0.0094656 , 0.00232344,
    0.00204926, 0.        , 0.00138449, 0.00556319, 0.00206143,
    0.        , 0.        , 0.00479916, 0.        , 0.        ,
    0.00940087, 0.00819726, 0.01408787, 0.00986389, 0.00705121,
    0.00710466, 0.009219  , 0.01365447, 0.00806681, 0.00919778,
    0.01522656, 0.0168779 , 0.01785582, 0.01872945, 0.01651353,
    0.01649871, 0.01083694, 0.01220166, 0.01132393, 0.01745626,
```

```
    0.01892118, 0.01657604, 0.01664315, 0.0158152 , 0.01810611,
    0.01774884, 0.01368326, 0.01732239, 0.01147137, 0.01394434,
    0.00624643, 0.00499297, 0.00814233, 0.00734853, 0.0127418 ,
    0.01142764, 0.01312612, 0.01043855, 0.01125843, 0.00560054,
    0.01719026, 0.01355098, 0.01216947, 0.01481042, 0.01594309,
    0.01813137, 0.00836498, 0.01495043, 0.01634219, 0.0091219 ,
    0.01447721, 0.01201847, 0.01156356, 0.00808543, 0.00649325,
    0.0099459 , 0.00725139, 0.01321795, 0.01504951, 0.01332466,
    0.00367928, 0.00338979, 0.00780954, 0.00909831, 0.01221079,
    0.00502451, 0.01245693, 0.00953705, 0.00316685, 0.00474482,
    0.01423495, 0.00683516, 0.0056664 , 0.0133458 , 0.00919483,
    0.01040229, 0.01117207, 0.0111454 , 0.00657364, 0.00781579])

c = np.array([
    0.63130104, 0.56263496, 0.60583358, 0.61347955, 0.62847238,
    0.56471462, 0.59810142, 0.56775415, 0.63595968, 0.58168552,
    0.72124287, 0.70925015, 0.74424832, 0.72173485, 0.68278937,
    0.70088118, 0.7295689 , 0.73810451, 0.76262451, 0.70812159,
    0.9955583 , 0.93923845, 0.93953618, 1.01383195, 1.01195795,
    0.97806329, 0.97489932, 0.94142746, 0.97362314, 0.96599846,
    0.23892097, 0.24531277, 0.28465395, 0.25744022, 0.21048498,
    0.21935477, 0.29602625, 0.2165602 , 0.23339493, 0.22058288,
    0.29069733, 0.25309897, 0.25847094, 0.33385684, 0.30967977,
    0.29942287, 0.27383582, 0.31358135, 0.29909066, 0.30447918,
    0.74512215, 0.78940189, 0.81717736, 0.79064375, 0.82609182,
    0.74383572, 0.81183109, 0.74202025, 0.81249634, 0.81090175,
    0.23704027, 0.22513036, 0.22963485, 0.25873413, 0.18113845,
    0.20531712, 0.18039346, 0.16676236, 0.25098528, 0.16510431,
    0.1777016 , 0.16465681, 0.18970901, 0.21730642, 0.20419822,
    0.21048861, 0.16149115, 0.16642746, 0.18080683, 0.18081711,
    0.16016021, 0.12865672, 0.10777411, 0.19073878, 0.15989937,
    0.15080329, 0.16159192, 0.15171408, 0.19052386, 0.17688598,
    0.95330452, 0.94840358, 0.91214702, 0.93137851, 0.9012103 ,
    0.95188041, 0.92656851, 0.99370568, 0.91058158, 0.92780834])

J_opt, S_opt = optimal_strategy(p, c, n)
print(S_opt)
```

3. Il suffit de considérer la stratégie optimale pour une unique loterie. Le gain total espéré associé à un support S est alors donné par

$$U(S) \triangleq g \sum_{j \in S} p_j - C(S) - \left(1 - \sum_{j \in S} p_j\right) \sum_{j \in S} c_j.$$

On observe immédiatement que l'ordre optimal des tickets reste celui induit par les ratios r_i. Soit $S = (s_1, \ldots, s_k)$ un support ordonné selon cet ordre. On a alors

$$U(S) - U(s_1, \ldots, s_k, s_{k+1}) = -g p_{s_{k+1}} + c_{s_{k+1}} \left(1 - \sum_{i=1}^{k} p_{s_i}\right),$$

$$U(S) - U(s_1, \ldots, s_{k-1}) = g p_{s_k} - c_{s_k} \left(1 - \sum_{i=1}^{k-1} p_{s_i}\right).$$

En divisant respectivement par $p_{s_{k+1}}$ et p_{s_k}, on obtient

$$\frac{U(S) - U(s_1, \ldots, s_k, s_{k+1})}{p_{s_{k+1}}} = -g + \frac{1 - \sum_{i=1}^{k} p_{s_i}}{r_{s_{k+1}}} \leq -g + \frac{1 - \sum_{i=1}^{k-1} p_{s_i}}{r_{s_{k+1}}},$$

$$\frac{U(S) - U(s_1, \ldots, s_{k-1})}{p_{s_k}} = g - \frac{1 - \sum_{i=1}^{k-1} p_{s_i}}{r_{s_k}}.$$

En additionnant ces deux inégalités, on obtient, dès que $r_{s_{k+1}} \geq r_{s_k}$,

$$\frac{U(S) - U(s_1, \ldots, s_k, s_{k+1})}{p_{s_{k+1}}} + \frac{U(S) - U(s_1, \ldots, s_{k-1})}{p_{s_k}} \leq 0.$$

Il en résulte que

$$\begin{aligned} U(S) &\leq \frac{p_{s_{k+1}} U(s_1, \ldots, s_{k-1}) + p_{s_k} U(s_1, \ldots, s_k, s_{k+1})}{p_{s_k} + p_{s_{k+1}}} \\ &\leq \max(U(s_1, \ldots, s_{k-1}), U(s_1, \ldots, s_k, s_{k+1})) \,. \end{aligned}$$

Ce résultat est analogue à celui du lemme établi à la question précédente et permet, par application du même théorème (en remplaçant J par U), de conclure que la stratégie optimale est obtenue en considérant un préfixe de l'ordre décroissant des ratios r_i.

Notes et références

La question 1 de cette énigme peut se reformuler de nombreuses façons équivalentes, selon le problème précis de recherche opérationnelle auquel on s'intéresse :

- **Search theory (théorie de la recherche)** : l'objectif est de minimiser le coût espéré avant de trouver un objet caché aléatoirement parmi n emplacements. Chaque emplacement est caractérisé par une probabilité connue de contenir l'objet et un coût d'exploration.
- **Troubleshooting (diagnostic séquentiel)** : de manière analogue, dans un système comportant plusieurs composants susceptibles d'être défaillants, le diagnostic consiste à tester séquentiellement ces composants afin d'identifier la panne. Chaque test est associé à un coût et à une probabilité de détecter la défaillance.
- **Single-machine scheduling (ordonnancement mono-machine)** : considérons n tâches à traiter sur une machine unique, chaque tâche étant définie par un temps de traitement et un poids représentant son importance. L'objectif est de minimiser la somme pondérée des temps de traitement.

Ce dernier domaine est sans doute le plus connu. Le résultat selon lequel l'ordre optimal est donné par le ratio p_i/c_i est appelé la *règle de Smith* [1]. Cette règle a été généralisée au cas où il existe des *contraintes de précédence* entre les tâches, à travers la notion de *décomposition de Sidney* [2]. Dans ce cadre, il convient d'explorer en priorité les « régions » A de densité $\rho(A) = \frac{\sum_{i \in A} p_i}{\sum_{i \in A} c_i}$ maximale.

La question 2, tirée de [3], est davantage liée à la théorie de la recherche. Elle peut être interprétée comme un problème de recherche séquentielle répétée avec possibilité d'abandon et de redémarrage.

Enfin, la question 3 correspond à un problème classique de *troubleshooting*, même si cela n'apparaît pas immédiatement. Plus précisément, il s'agit d'une variante dans laquelle une action supplémentaire, appelée *call service* [4], permet de réparer automatiquement le système moyennant un coût donné. Dans notre cas, au lieu de chercher à maximiser U, on peut chercher à minimiser $g - U$ (ce qui est équivalent). Le paramètre g peut alors être interprété comme le coût du *call service*, et $g - U$ comme le coût total espéré de réparation du système.

Sources

[1] Wayne E Smith et al. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[2] Jeffrey B Sidney. Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research*, 23(2):283–298, 1975.

[3] Pierre Perrault, Vianney Perchet, and Michal Valko. Finding the bandit in a graph : Sequential search-and-stop. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1668–1677. PMLR, 2019.

[4] David Heckerman, John S Breese, and Koos Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–57, 1995.