Empilement optimisé de sphères à rayons variables

appliqué au jeu de Pastèque

Antoine Salomon (Candidat 26580)

TIPE Session 2024

Introduction

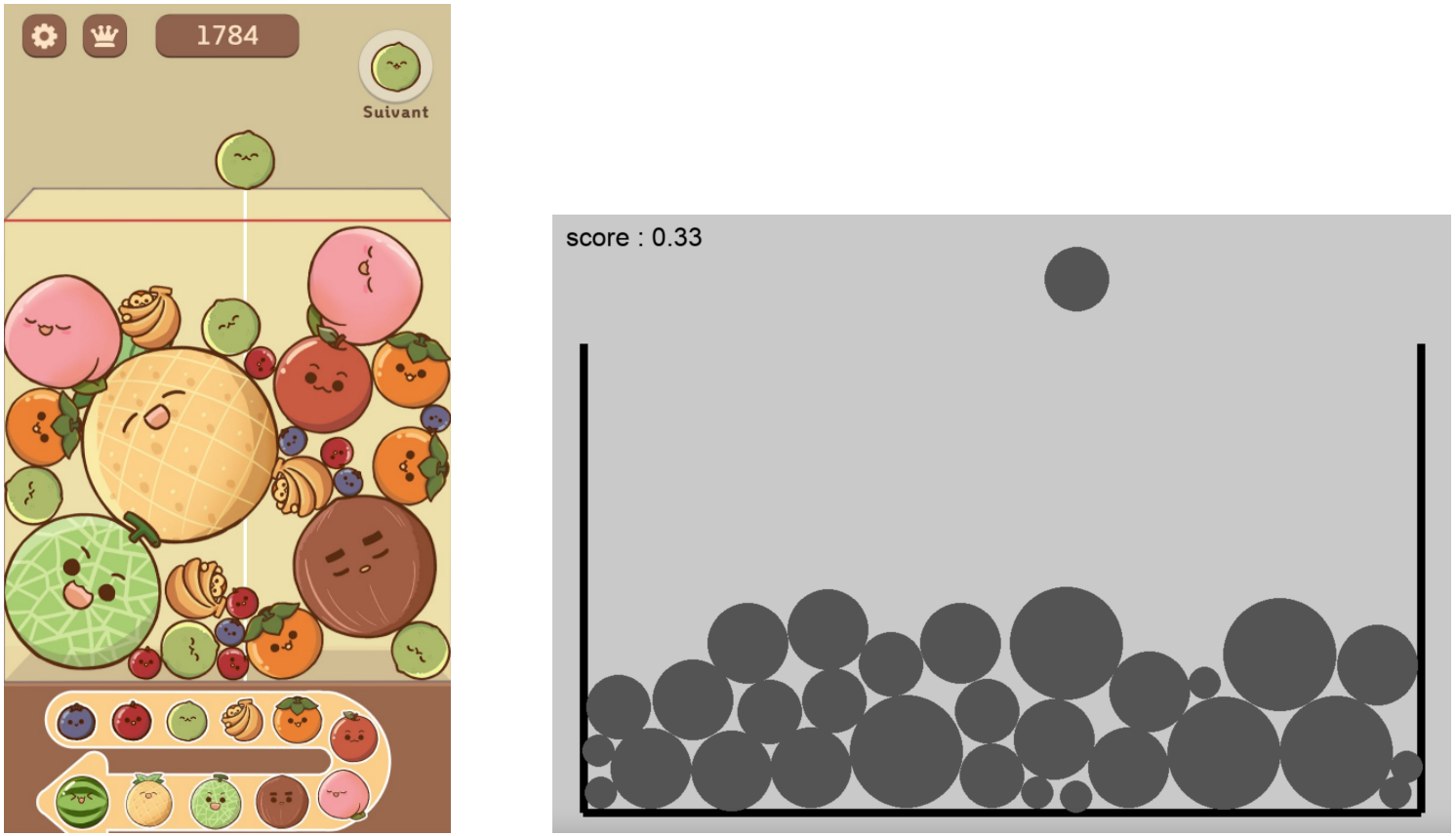

Figure: Jeu original (gauche) et mon implémentation (droite)

Problématique

Comment maximiser son score au jeu de Pastèque en optimisant la densité de l'empilement de sphères compactes de rayons différents ?

Plan de l'exposé

1. Recherche théorique d'un empilement optimisé
2. Application de cette approche à l'automatisation du jeu de Pastèque

Partie 1

Recherche théorique d'un empilement optimisé

Définition des constantes

- 4 rayons possibles : $R_1 < R_2 < R_3 < R_4$
- dimensions du cadre : a sa longueur et b sa hauteur
- n un entier vérifiant

$$n\pi R_1^2 > ab$$

qui correspond au nombre maximal de sphères utilisées dans un empilement valide

Idées directrices

Se ramener à l'étude d'un nombre fini d'emplacements possibles pour le centre des sphères d'un empilement

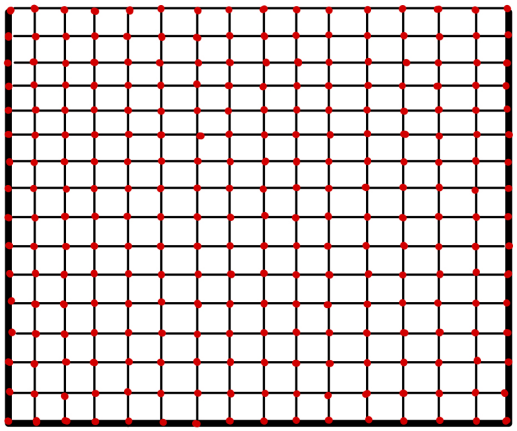

Figure: Grille d'ordre 16 (16×16 emplacements)

On note l'ensemble des empilements valides sur la grille d'ordre m CG(m)

Idées directrices

On peut s'approcher autant que l'on veut d'une densité optimale en restant sur la grille, ou plus formellement

$$\forall \varepsilon > 0, \exists m \in \mathbb{N}, max[d(c), c \in CG(m)] > d_{max} - \varepsilon \qquad (1)$$

où d_{max} est la densité maximale atteinte pour un empilement de sphères de rayons R_1, R_2, R_3 ou R_4

Idée de la preuve

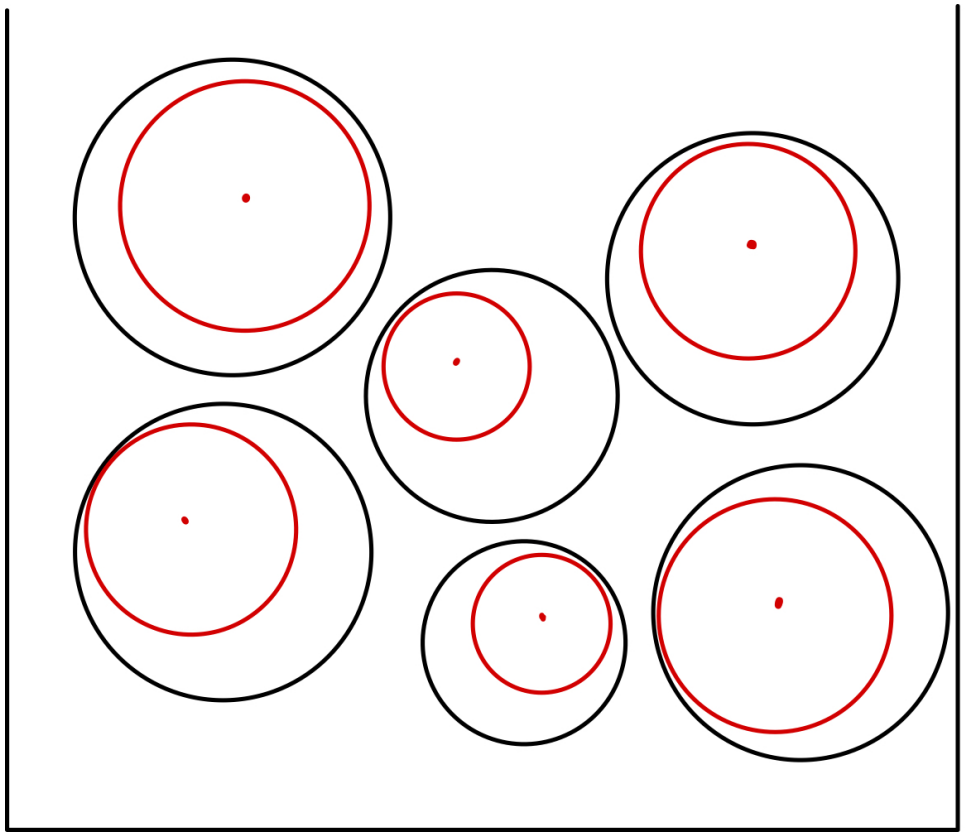

Figure: Empilement sur la grille se rapprochant d'une densité optimale

Obtention algorithmique de l'empilement optimisé

- Algorithme Séparer et Évaluer

```
def heuristique(g, i_actuel, j_actuel) :
    pas = (1035 - 18) / n
    largeur_actuelle = j_actuel * pas
    hauteur_actuelle = i_actuel * pas
    return (densite(g) - 1) * largeur_actuelle * hauteur_actuelle + a * b
```

Figure: Heuristique

- On obtient finalement la configuration sous forme d'un tableau de dimension $m \times m$

Partie 2

Application de cette approche à l'automatisation du jeu de Pastèque

Stratégie optimisée

Algorithm 1 Principe de l'automatisation

Soit g la grille décrivant l'état de la partie
Soit g_{opt} l'empilement optimisé construit à partir de g
for i = 0 to len(g_{opt}) **do**
    for j = 0 to len(g_{opt}) **do**
        if $g_{opt}[i][j]$ est égal au rayon actuel **then**
            return abscisse correspondant
        end if
    end for
end for
return abscisse aléatoire

Les autres stratégies

- Stratégie aléatoire
- Stratégie gloutonne

Empilements obtenus

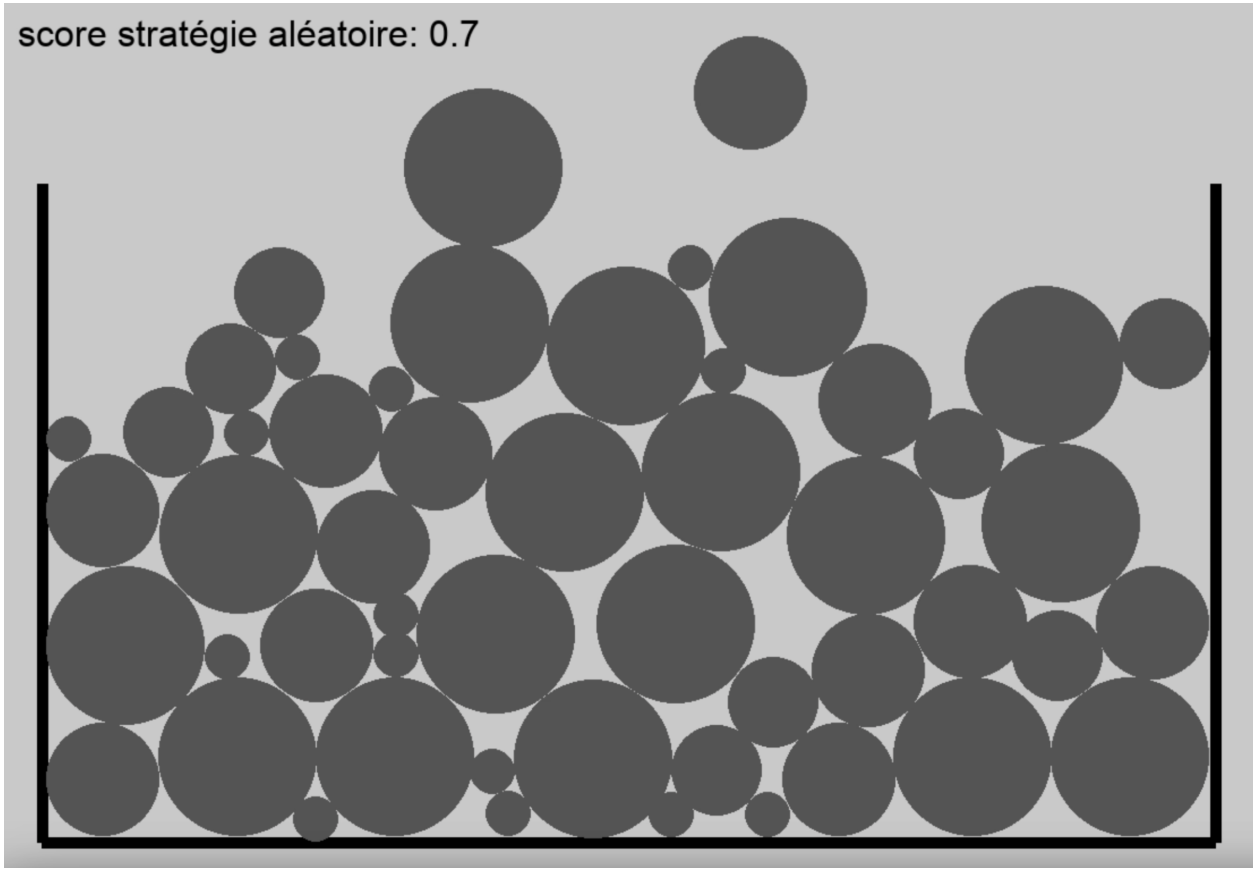

Figure: Partie stratégie aléatoire

Empilements obtenus

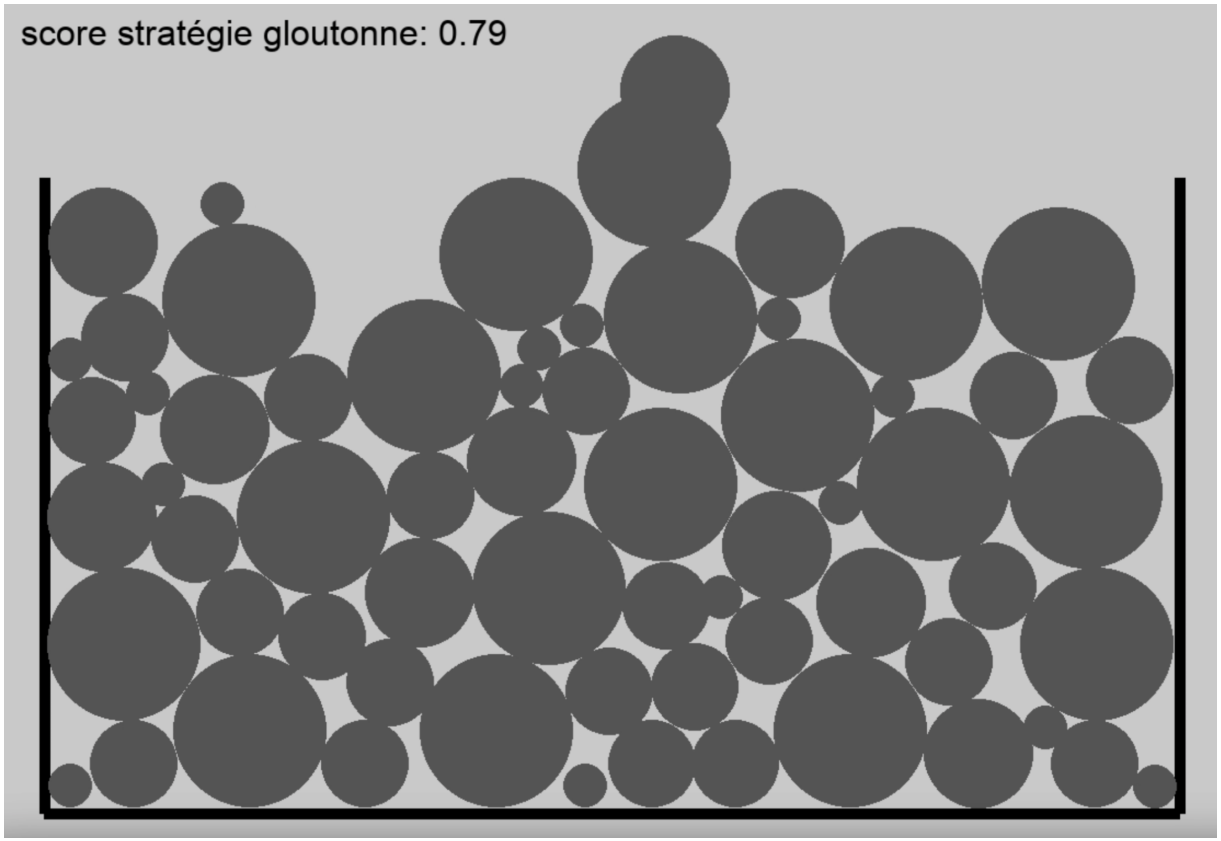

Figure: Partie stratégie gloutonne

Empilements obtenus

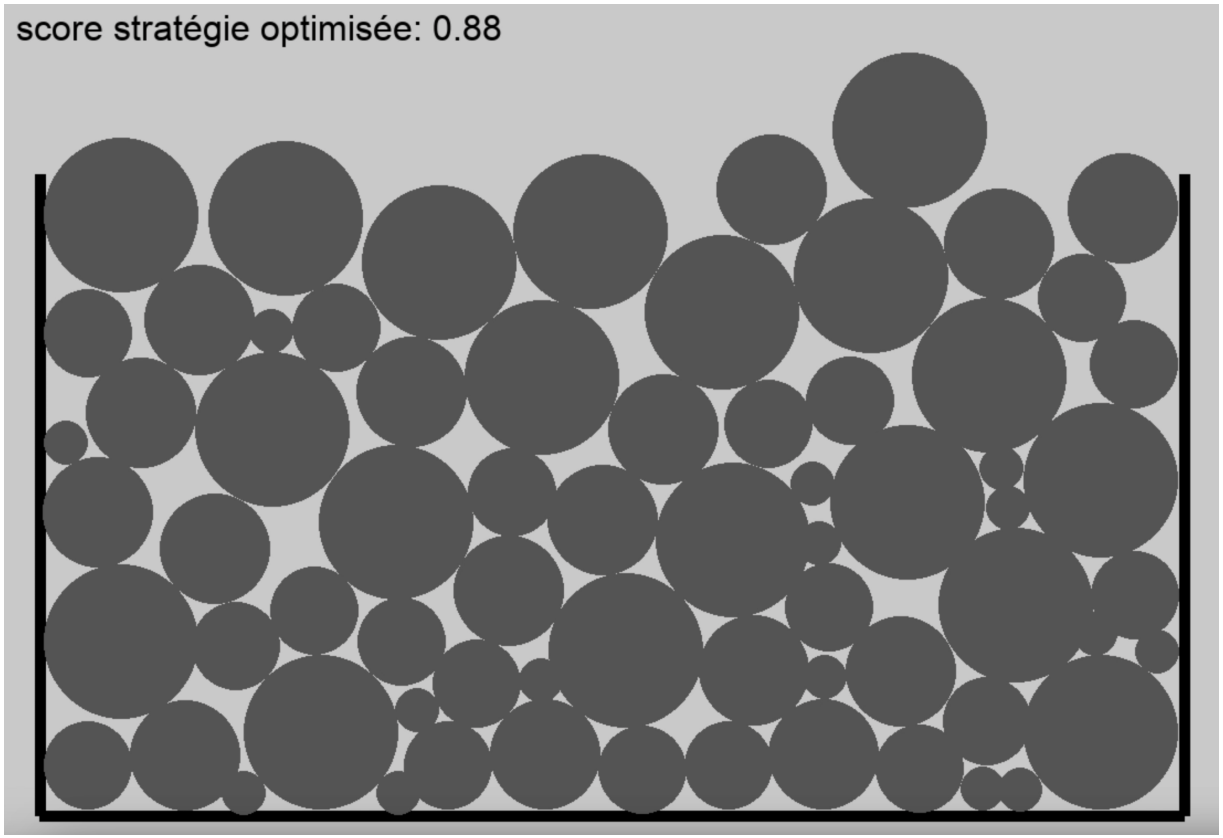

Figure: Partie stratégie optimisée

Résultats obtenus

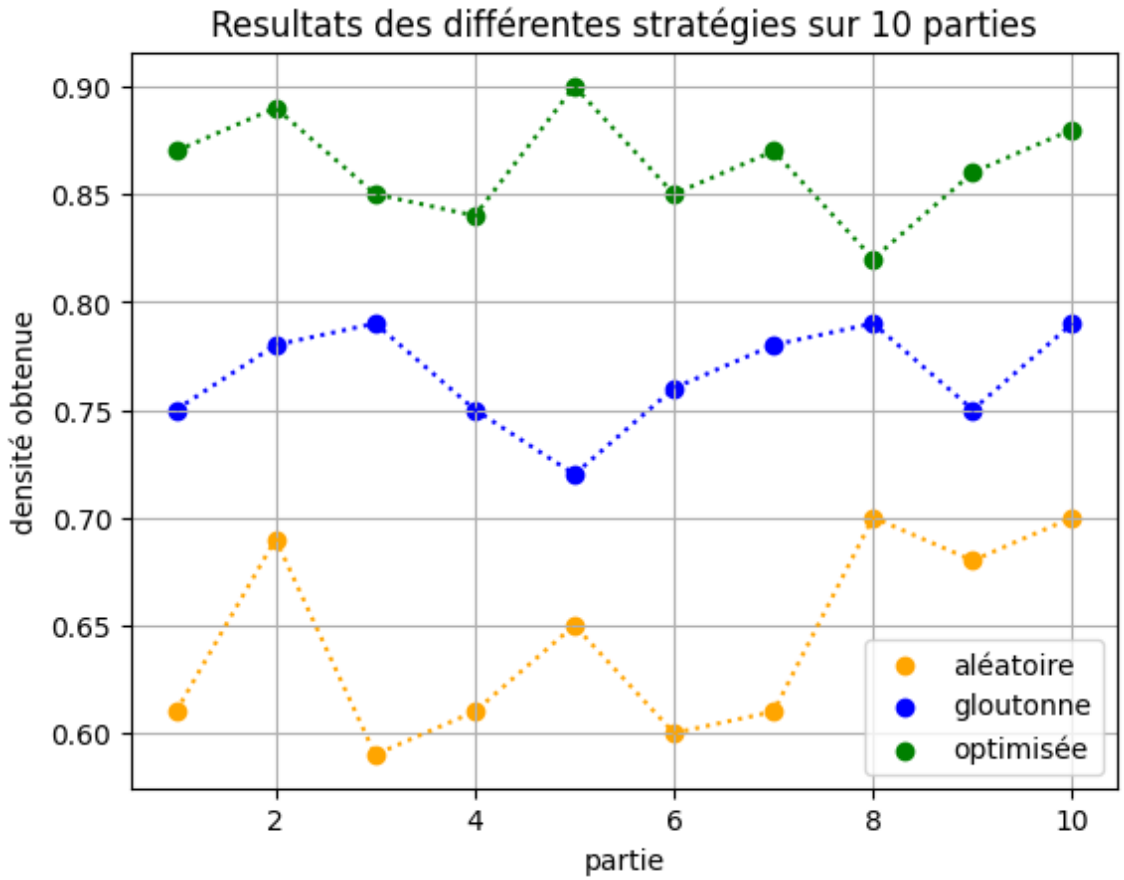

Figure: Résultats sur 10 parties

Conclusion et ouverture

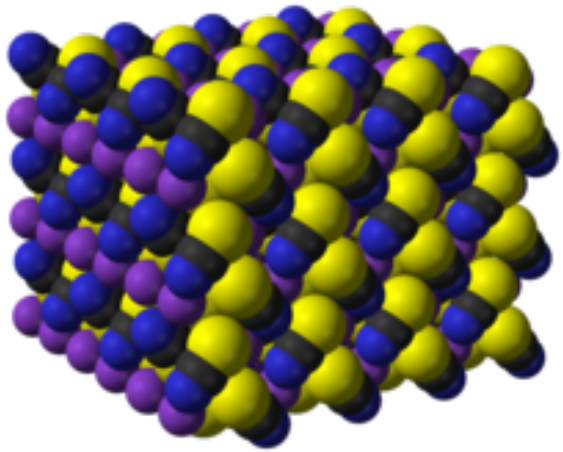

Figure: Empilement de sphères en cristallographie

Merci beaucoup pour votre attention

Annexes

Annexe : Preuve de la proposition (1)

- On admet l'existence d'un empilement c_0 valide de sphères de rayons $\frac{R_1}{\rho_m}$, $\frac{R_2}{\rho_m}$, $\frac{R_3}{\rho_m}$ et $\frac{R_4}{\rho_m}$ (où ρ_m est un coefficient défini en annexe) de densité maximale d_0 pour ces rayons
- On réduit le rayon des sphères de l'empilement c_0 et on les décale sur des points de la grille

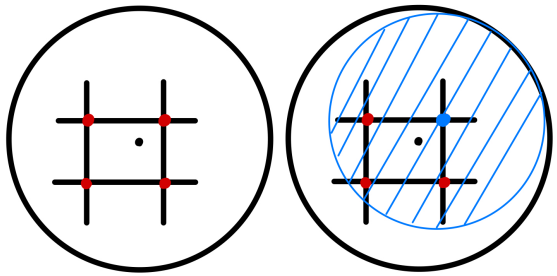

Figure: Illustration du décalage d'une sphère

Annexe : Preuve de la proposition (1)

Majoration de la perte

- La perte due à la réduction des rayons est majorée par $p_m = \frac{n\pi \sum_1^4 R_k^2 (\frac{1}{\rho_m^2} - 1)}{ab}$
- La différence entre la densité maximale avec les rayons initiaux et celle avec les nouveaux rayons est majorée par $d_0(\rho_m^2 - 1) + 1 - \rho_m^2$ car $d_{max} < d_0\rho_m^2 + (1 - \rho_m)(1 + \rho_m)$

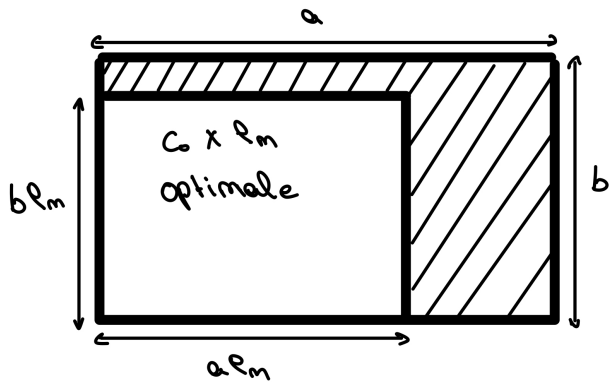

Figure: Idée pour la majoration

Annexe : Preuve de la proposition (1)

- Majoration de la distance au point de la grille le plus proche : $e_m = \frac{1}{m} \times \sqrt{a^2 + b^2}$
- Coefficient de réduction des rayons $\rho_m = min_{k=1}^{4}(\frac{R_k}{R_k + 2e_m})$ de sorte que $R_k \leq \frac{R_k}{\rho_m} - 2e_m$

```
1  def e(m) :
2      return math.sqrt(a*a + b*b) / (m)          # deplacement maximal des centres pour les placer sur la grille
3
4  def rho(m) :
5      min = (rayons[0]/(rayons[0] + e(m)))
6      for k in range (len(rayons)) :
7          if ((rayons[k]/(rayons[k] + e(m))) < min ) :
8              min = (rayons[k]/(rayons[k] + e(m)))
9      return min
10
11
12
13
14
15
16
17
```

```
18 def p(m) :
19     res = 0
20     for k in range(len(rayons)) :
21         res += rayons[k] * rayons[k] * (1/(rho(m
       )*rho(m)) - 1)
22     return n * math.pi * res
23
24 def majoration_perte(m) :
25     return (1 - rho(m) * rho(m)) * (1 - dmin) +
       (p(m) / (a * b))
26
27 def taille_grille_pour_seuil(epsilon) :
28     m = 1
29     while (majoration_perte(m) > epsilon) :
30         m += 1
31     return m
```

Listing 1: Annexe pour la taille de la grille

```
1 def configuration_optimisee(g, i_actuel,
      j_actuel, densite_opt, grille_opt) :
```

```
    if i_actuel == len(g) and j_actuel == len(g)
    and densite(g) > densite_opt:
        grille_opt = g
        densite_opt = densite(g)
    if g[i_actuel][j_actuel] >= 0 :
        for r in range(0, 4) :
            new_i, new_j = i_actuel, j_actuel
            incr_indice(new_i, new_j, g)
            g[i_actuel][j_actuel] = r
            if est_valide(g) and heuristique(g,
i_actuel, j_actuel) > densite_opt :
                configuration_optimisee(g, new_i
, new_j, densite_opt, grille_opt)
    else :
        incr_indice(i_actuel, j_actuel, g)
        configuration_optimisee(g, i_actuel,
j_actuel, densite_opt, grille_opt)
    return grille_opt
```

Listing 2: Annexe pour l'obtention de l'empilement optimitsé

```
1  def grille_etat_partie(m, sph) :
2      g = [[0 for i in range(0, m)] for j in range
       (0, m)]
3      pas_h = largeur / m
4      pas_v = hauteur / m
5      for s in sph :
6          i, j = 0, 0
7          while s.pos[0] > (j + 1) * pas_h:
8              j += 1
9          while s.pos[1] > (i + 1) * pas_v:
10             i += 1
11         g[i][j] = - ind_rayon(rayons, s.rayon)
12     return g
13
14
15
16 def jouer_coup_glouton(n, sph) :
17     pas = (1035 - 18) / n
18     g = grille_etat_partie(n, sph)
19     hauteur_colonnes = [0 for i in range(n)]
```

```
20      for j in range(n) :
21          for i in range(n) :
22              if g[i][j] != 0 :
23                  hauteur_colonnes[j] += rayons
    [-1-g[i][j]]
24      ind_min = 0
25      print(hauteur_colonnes)
26      for i in range(len(hauteur_colonnes)) :
27          if hauteur_colonnes[i] <
    hauteur_colonnes[ind_min] :
28              ind_min = i
29      print(ind_min)
30      return pas * ind_min
```

Listing 3: Annexe pour la stratégie gloutonne