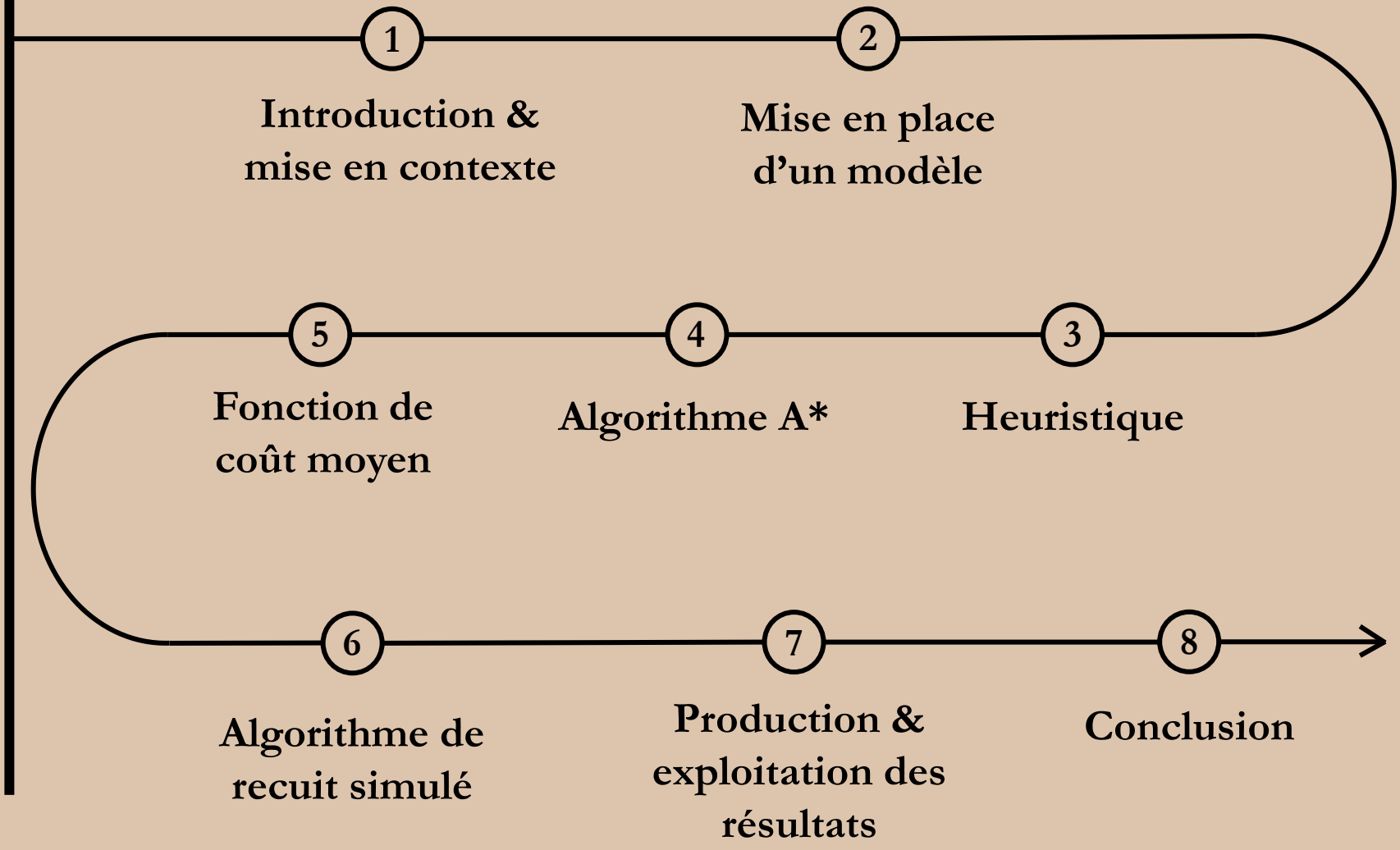
TIPE – Optimisation de l'occupation des places d'un parking automatisé

Candidat n°34769
DEBES Maxence

Sommaire


1
Introduction & mise en contexte
2
Mise en place d'un modèle
3
Heuristique
4
Algorithme A*
5
Fonction de coût moyen
6
Algorithme de recuit simulé
7
Production & exploitation des résultats
8
Conclusion

1 – Introduction & mise en contexte

- **Les parkings automatisés : des avantages considérables**

- **Un problème d'optimisation ?**

Exemple d'un parking automatisé sur plusieurs étages – Source : www.ornikar.com/permis/conseils-conduite/stationnement/amenagements/parkings-automatises

2 – Mise en place d'un modèle

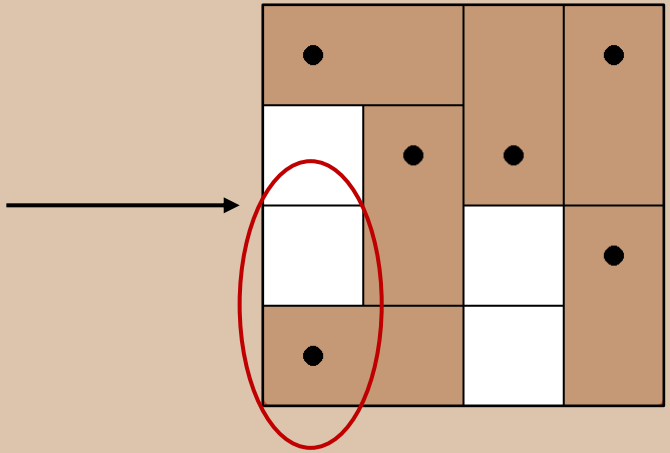
```
>>> [(n, p), k, [(a1, b1), (a2, b2), ..., (ak, bk)]]
```

Nombre de lignes et de colonnes du parking

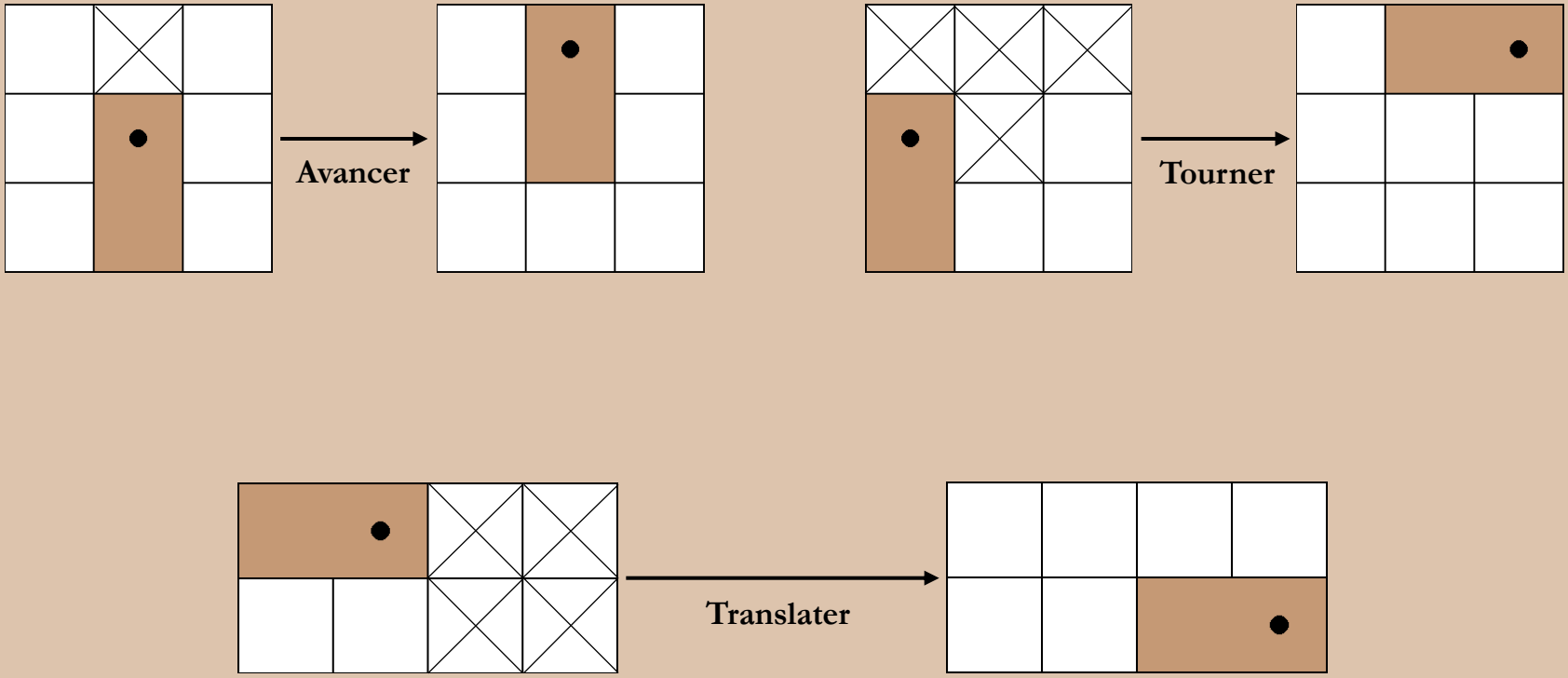
Nombre de voitures dans le parking

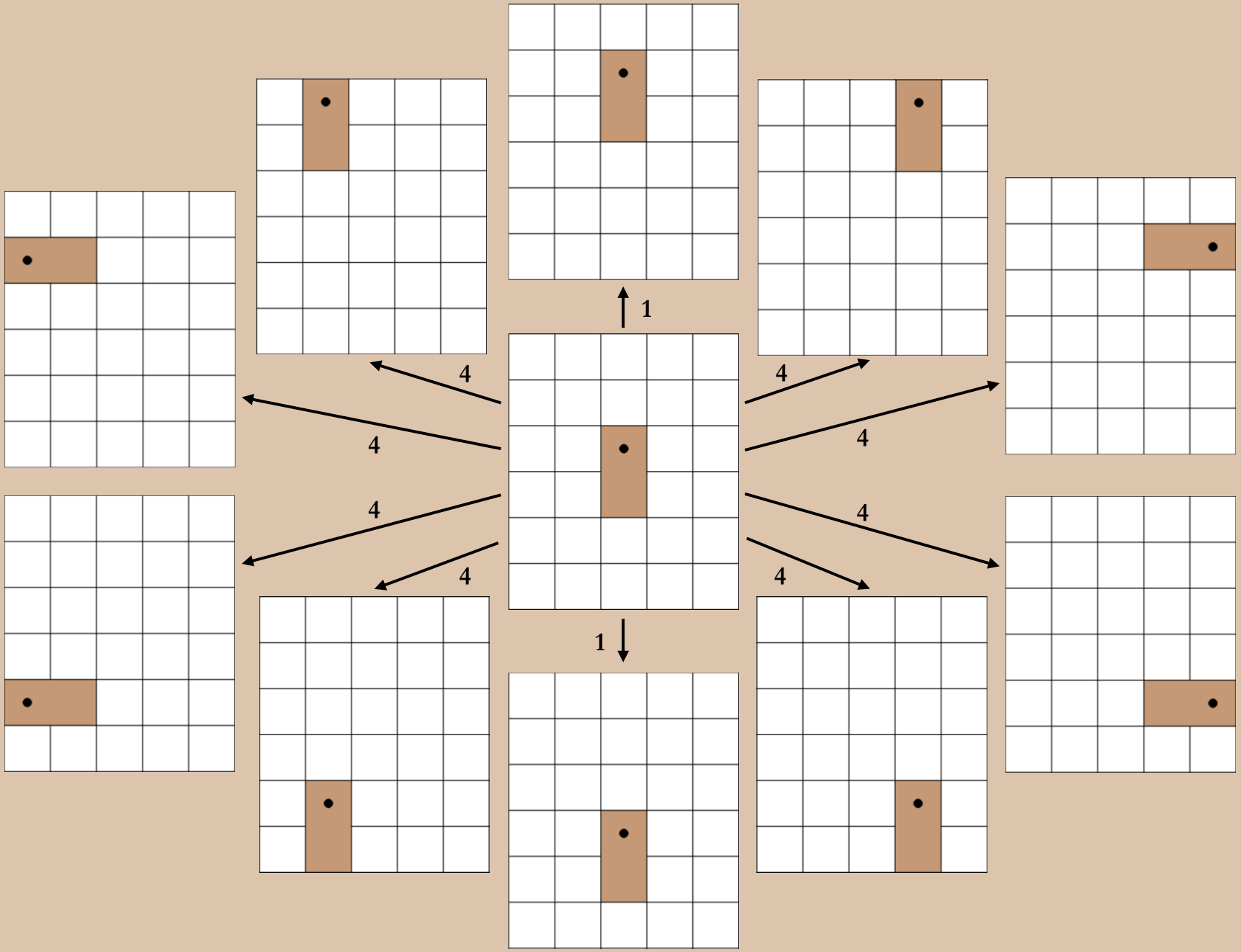
Liste des emplacements des voitures dans le parking, le premier élément de chaque couple représentant l'avant de la voiture

```
>>> creer_parking([(4, 4), 6, [(1, 2), (4, 8),
(12, 16), (13, 14), (6, 10), (7, 3)]])
array([[2., 1., 1., 2.],
       [0., 2., 2., 1.],
       [0., 1., 0., 2.],
       [2., 1., 0., 1.]])
```

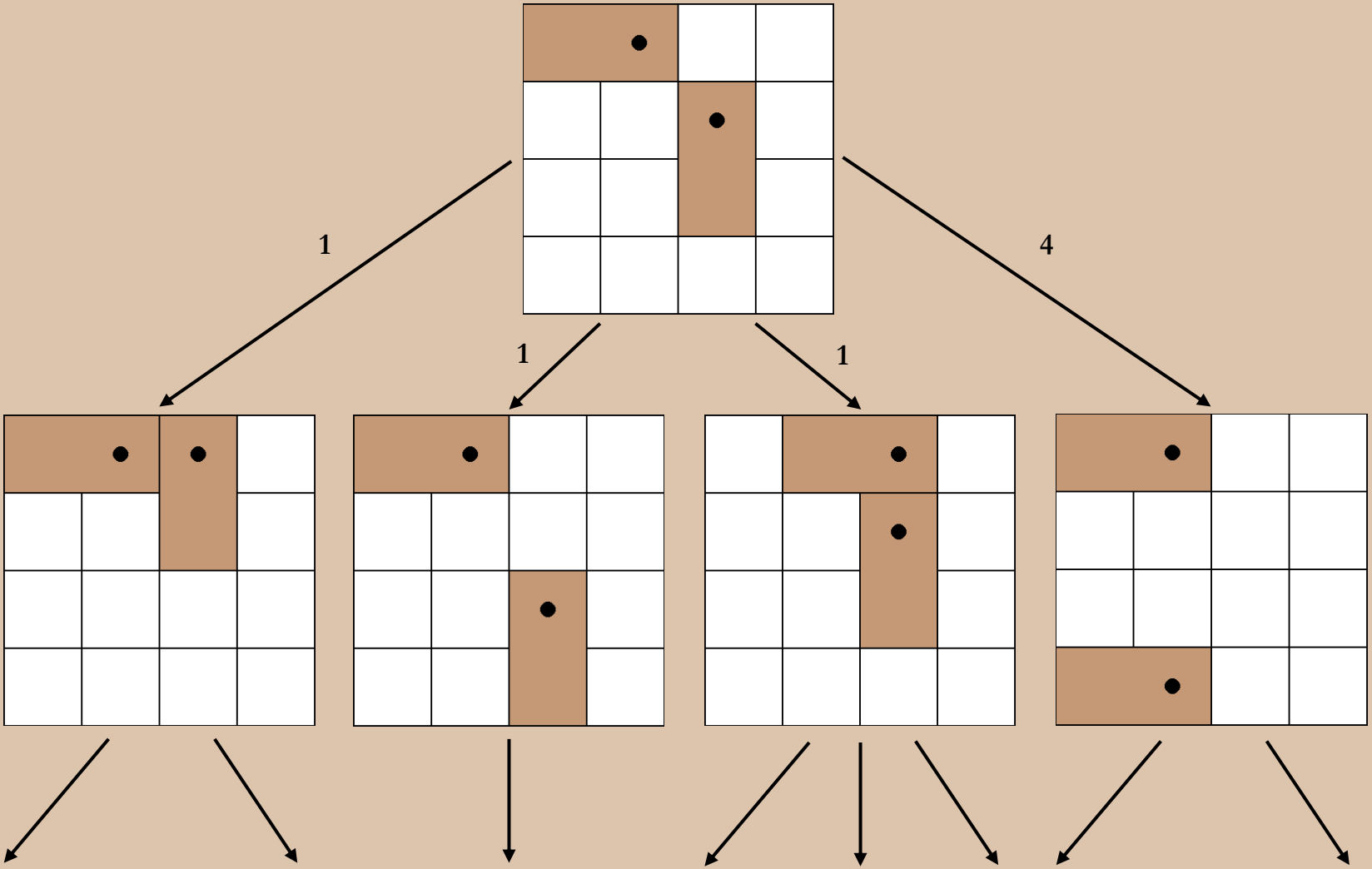


Sortie du parking


Avancer
Tourner
Translater


1
4
4
4
4
4
4
4
4
1

Structure arborescente :



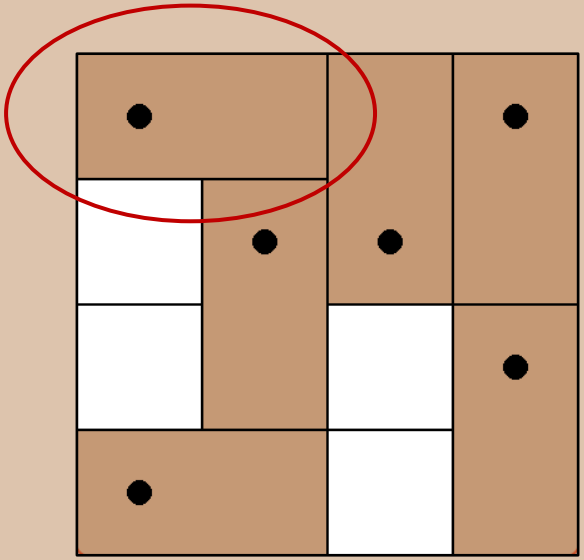
3 – Heuristique

Deux qualités :

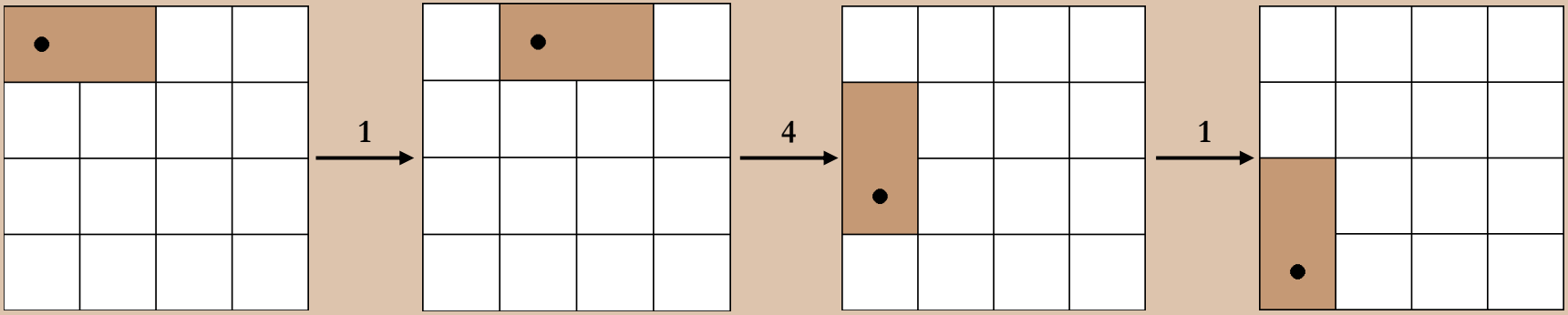
- **Admissible : ne surestime jamais le coût pour se rendre à l'objectif**

- **Cohérente : pour tout nœud N, pour tout successeur S de N, $h(N) \leq cout(N,S) + h(S)$**

Cohérente $\Longrightarrow$ **Admissible**



```
>>> heuristique([(4, 4), 6, [(1, 2), (4, 8),
(12, 16), (13, 14), (6, 10), (7, 3)]], 0)
6
```



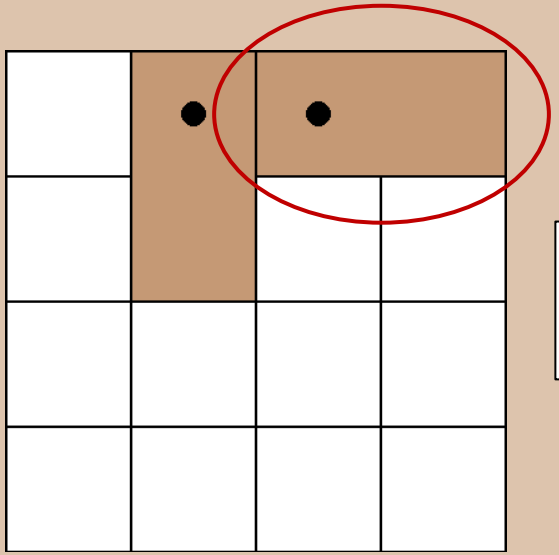
4 – Algorithme A*

Noeud	N1	N2	N3	…	Nk
Valeur selon f	2	7	11	…	19

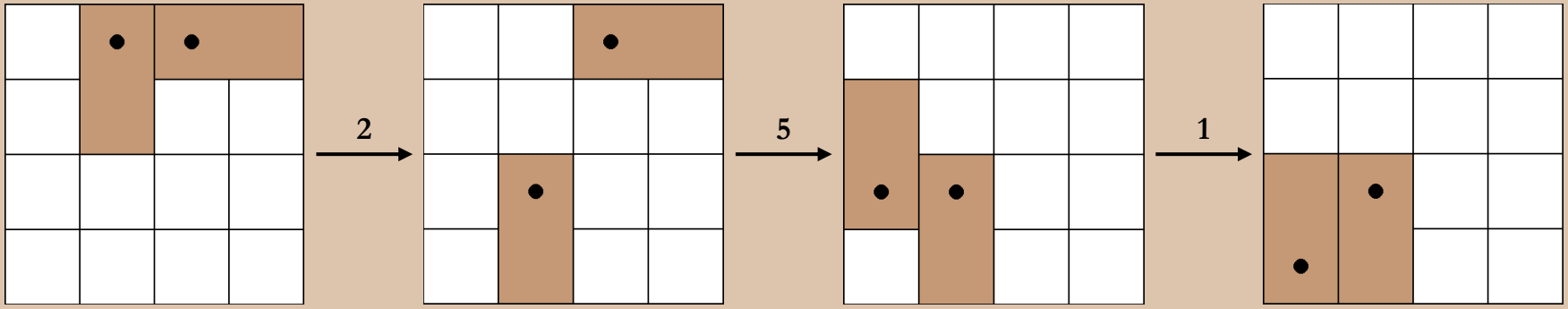
$$f(N) = g(N) + h(N)$$

Supposons que N1 ait deux fils N4 et N5 tels que f(N4) = 5 et f(N5) = 9 :

Noeud	N4	N2	N5	N3	…	Nk
Valeur selon f	5	7	9	11	…	19

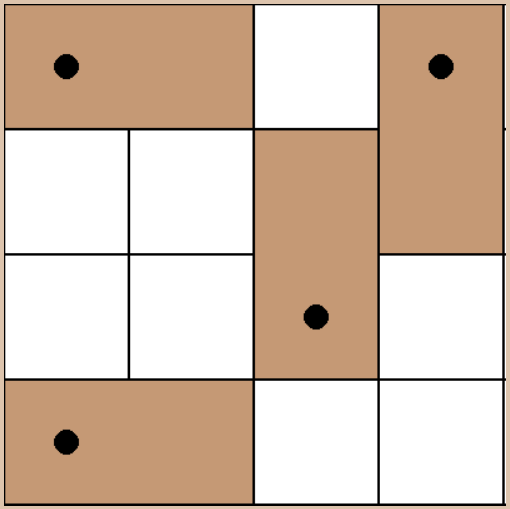


```
>>> algorithme_A_etoile([(4, 4), 2, [(2, 6),
(3, 4)]], (3, 4))
8
```



5 – Fonction de coût moyen

- Calcul du coût optimisé grâce à l'algorithme A*



```
>>> cout_moyen([(4, 4), 4, [(1, 2), (4, 8),
(13, 14), (11, 7)]])
10.5
```

- Recherche d'une disposition optimale ?

6 – Algorithme de recuit simulé

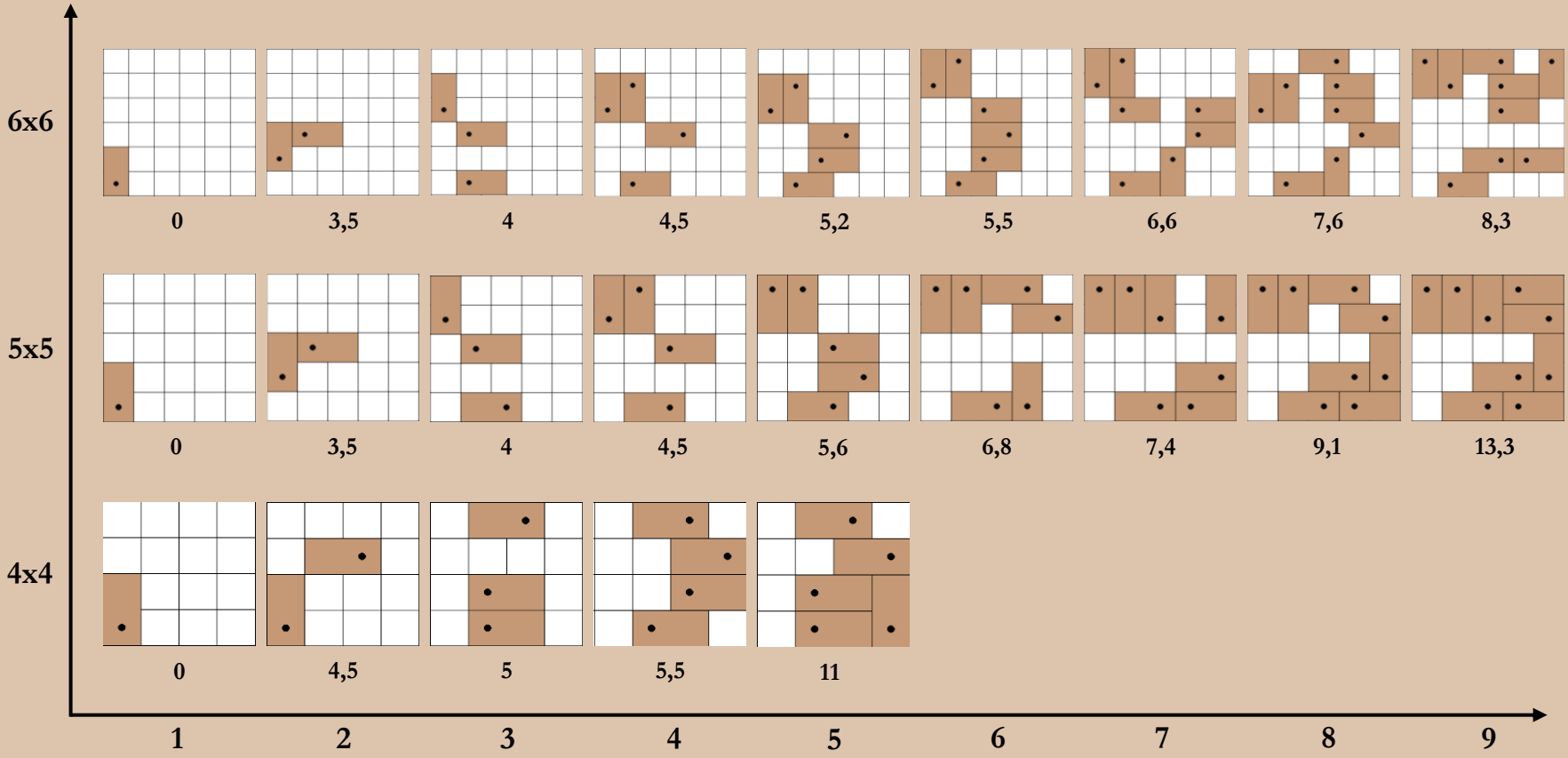
- Recherche d'un minimum global via une métaheuristique

- Principe de l'algorithme : décroissance de la température de façon continue, critère d'acceptation d'un voisin généré aléatoirement (règle de Metropolis)

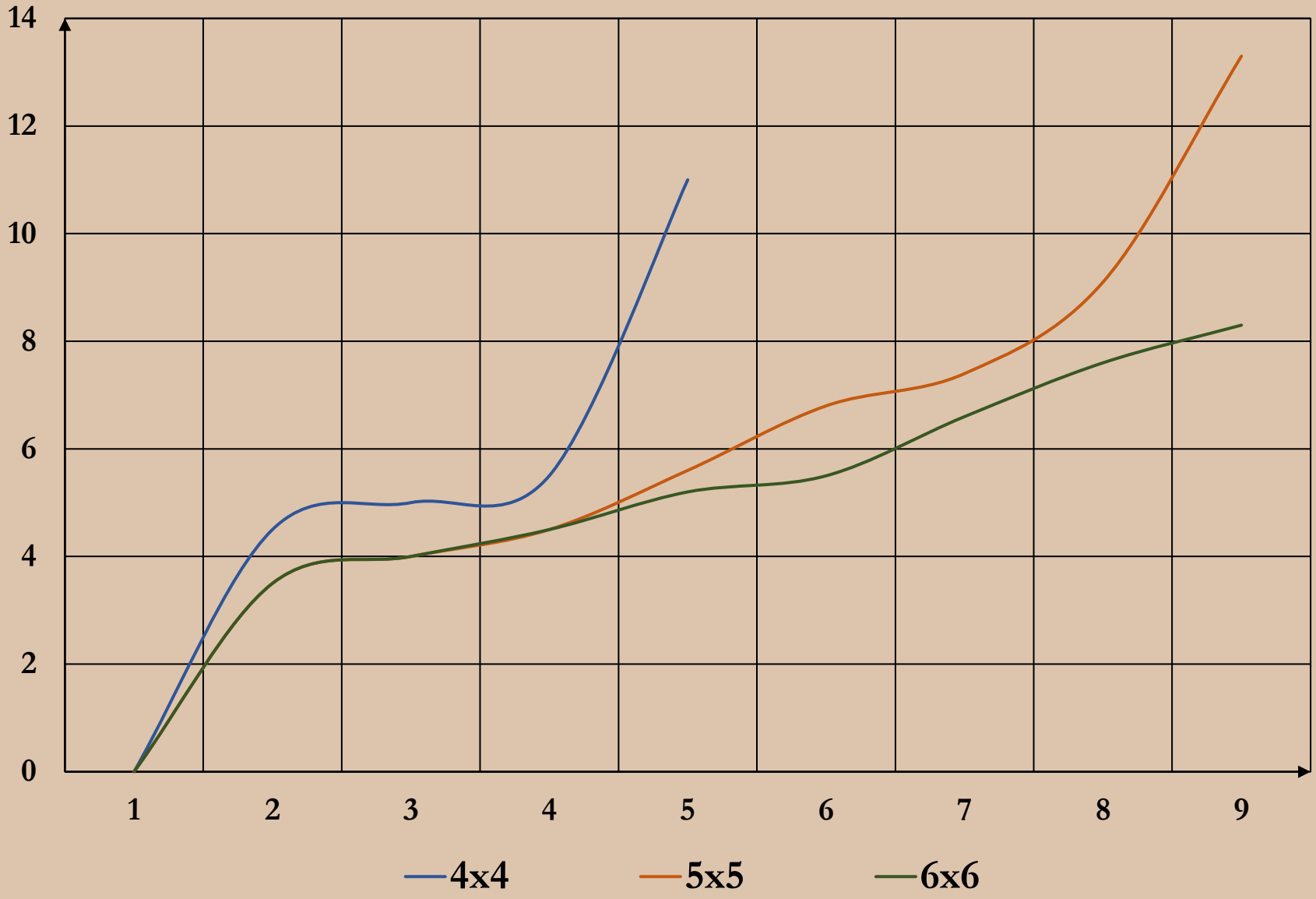
```
        # Évaluation des solutions
        cout_actuel = cout_moyen(Noeud_actuel)
        cout_voisin = cout_moyen(Voisin)
        delta_cout = cout_voisin - cout_actuel
        if delta_cout < 0:
            Noeud_actuel = Voisin
            compteur_stagnation = 0
        elif delta_cout == 0:
            compteur_stagnation += 1
        else:
            if np.exp(-delta_cout/temperature) > random.uniform(0,1):
                Noeud_actuel = Voisin
                compteur_stagnation = 0
            else:
                compteur_stagnation += 1
```

Intérêt de l'acceptation d'un « mauvais » état

7 – Production & exploitation des résultats



Coût moyen en fonction du nombre de voitures



8 – Conclusion

- **Des résultats concluants, à quel prix ?**

- **Piste d'amélioration : mémoïsation**

```python
# TIPE.py

import numpy as np
import pygame
import random
import copy

def creer_parking(Noeud):

    # Initialisation des données
    dimensions, k, Emplacements_Voitures = Noeud
    n, p = dimensions
    P = np.zeros((n, p))

    # Placement des voitures
    for Voiture in Emplacements_Voitures:
        a, b = Voiture
        q1, r1 = divmod(a, p)
        q2, r2 = divmod(b, p)

        # Avant de la voiture
        if r1 == 0:
            P[q1-1][p-1] = 2
        else:
            P[q1][r1-1] = 2

        # Arrière de la voiture
        if r2 == 0:
            P[q2-1][p-1] = 1
        else:
            P[q2][r2-1] = 1

    return P
```

```python
def dessiner_parking(Noeud):

    # Initialisation des données
    dimensions, nb_voitures, Emplacements_voitures = Noeud
    n, p = dimensions

    # Initialisation de pygame
    pygame.init()

    # Définition des dimensions de la fenêtre
    largeur_fenetre = p * 100
    hauteur_fenetre = n * 100
    taille_fenetre = (largeur_fenetre, hauteur_fenetre)

    # Création de la fenêtre
    fenetre = pygame.display.set_mode(taille_fenetre)

    # Définition des couleurs
    couleur_case = (255, 255, 255)
    couleur_bord = (0, 0, 0)
    couleur_voiture = (197, 153, 117)
    couleur_symbole = (0, 0, 0)

    while True:
        # Gestion des événements
        for evenement in pygame.event.get():
            if evenement.type == pygame.QUIT:
                pygame.quit()
                return

        # Effacement de la fenêtre
        fenetre.fill(couleur_bord)
```



```python
        # Dessin de la grille
        for i in range(n):
            for j in range(p):
                pygame.draw.rect(fenetre, couleur_case, (j*100+1, i*100+1, 98, 98))

        # Dessin des bordures
        for i in range(n+1):
            pygame.draw.line(fenetre, couleur_bord, (0, i*100), (largeur_fenetre,
i*100), 1)
        for i in range(p+1):
            pygame.draw.line(fenetre, couleur_bord, (i*100, 0), (i*100,
hauteur_fenetre), 1)

        # Dessin des voitures
        for voiture in Emplacements_voitures:
            a, b = voiture
            q1, r1 = divmod(a-1, p)
            q2, r2 = divmod(b-1, p)

            # Détermination du sens de la voiture
            if r1 == r2: # voiture verticale
                debut, fin = min(q1, q2), max(q1, q2)
                pygame.draw.rect(fenetre, couleur_voiture, (r1*100+1, debut*100+1,
98, 100))
                pygame.draw.rect(fenetre, couleur_voiture, (r1*100+1,
(debut+1)*100+1, 98, 98))
            else: # voiture horizontale
                debut, fin = min(r1, r2), max(r1, r2)
                pygame.draw.rect(fenetre, couleur_voiture, (debut*100+1, q1*100+1,
100, 98))
                pygame.draw.rect(fenetre, couleur_voiture, ((debut+1)*100+1,
q1*100+1, 98, 98))

            # Dessin du symbole sur la case représentant le devant de la voiture
            pygame.draw.circle(fenetre, couleur_symbole, ((r1*100)+50, (q1*100)+50),
10)

        # Actualisation de la fenêtre
        pygame.display.flip()
```

```python
def enfants_noeud(Noeud):

    # Initialisation des données
    dimensions, k, Emplacements_voitures = Noeud
    n, p = dimensions
    Enfants = []
    P = creer_parking(Noeud)

    # Avancer
    for i in range (len(Emplacements_voitures)):
        a, b = Emplacements_voitures[i]
        q1, r1 = divmod(a, p)
        q2, r2 = divmod(b, p)

        # Voiture horizontale
        if a == b-1:
            if r1 != 1:
                if P[q1][r1-2] == 0:
                    S = copy.deepcopy(Noeud)
                    S[2][i] = (a-1, b-1)
                    Enfants.append(S)
        if a == b+1:
            if r1 != 0:
                if P[q1][r1] == 0:
                    S = copy.deepcopy(Noeud)
```

```
                S[2][i] = (a+1, b+1)
                Enfants.append(S)

        # Voiture verticale
        if a == b-p:
            if r1 == 0:
                if q1 != 1:
                    if P[q1-2][p-1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-p, b-p)
                        Enfants.append(S)
            else:
                if q1 != 0:
                    if P[q1-1][r1-1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-p, b-p)
                        Enfants.append(S)
        if a == b+p:
            if r1 == 0:
                if q1 != n:
                    if P[q1][p-1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+p, b+p)
                        Enfants.append(S)
            else:
                if q1 != n-1:
                    if P[q1+1][r1-1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+p, b+p)
                        Enfants.append(S)

    # Reculer
    for i in range (len(Emplacements_voitures)):
        a, b = Emplacements_voitures[i]
        q1, r1 = divmod(a, p)
        q2, r2 = divmod(b, p)

        # Voiture horizontale
        if a == b-1:
            if r2 == p-1:
                if P[q2][p-1] == 0:
                    S = copy.deepcopy(Noeud)
                    S[2][i] = (a+1, b+1)
                    Enfants.append(S)
            elif r2 != 0:
                if P[q2][r2] == 0:
                    S = copy.deepcopy(Noeud)
                    S[2][i] = (a+1, b+1)
                    Enfants.append(S)
        if a == b+1:
            if r2 != 1:
                if P[q2][r2-2] == 0:
                    S = copy.deepcopy(Noeud)
                    S[2][i] = (a-1, b-1)
                    Enfants.append(S)

        # Voiture verticale
        if a == b-p:
            if r2 == 0:
                if q2 != n:
                    if P[q2][p-1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+p, b+p)
                        Enfants.append(S)

            else:
                if q2 != n-1:
```



```
                    if P[q2+1][r2-1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+p, b+p)
                        Enfants.append(S)
            if a == b+p:
                if r2 == 0:
                    if q2 != 1:
                        if P[q2-2][p-1] == 0:
                            S = copy.deepcopy(Noeud)
                            S[2][i] = (a-p, b-p)
                            Enfants.append(S)
                else:
                    if q2 != 0:
                        if P[q2-1][r2-1] == 0:
                            S = copy.deepcopy(Noeud)
                            S[2][i] = (a-p, b-p)
                            Enfants.append(S)

        # Tourner avant droit
        for i in range (len(Emplacements_voitures)):
            a, b = Emplacements_voitures[i]
            q1, r1 = divmod(a, p)
            q2, r2 = divmod(b, p)

            # Voiture horizontale
            if a == b-1:
                if r1 != 1 :
                    if q1 >= 2:
                        if P[q1-1][r1-1] == 0 and P[q1][r1-2] == 0 and P[q1-1][r1-2] == 0
and P[q1-2][r1-2] == 0:
                            S = copy.deepcopy(Noeud)
                            S[2][i] = (a-1-2*p, b-2-p)
                            Enfants.append(S)

            if a == b+1:
                if r1 != 0:
                    if q1 <= n-3:
                        if P[q1+1][r1-1] == 0 and P[q1][r1] == 0 and P[q1+1][r1] == 0 and
P[q1+2][r1] == 0:
                            S = copy.deepcopy(Noeud)
                            S[2][i] = (a+1+2*p, b+2+p)
                            Enfants.append(S)

            # Voiture verticale
            if a == b-p:
                if q1 >= 1:
                    if 1 <= r1 <= p-2:
                        if P[q1][r1] == 0 and P[q1-1][r1-1] == 0 and P[q1-1][r1] == 0 and
P[q1-1][r1+1] == 0:
                            S = copy.deepcopy(Noeud)
                            S[2][i] = (a-p+2, b-2*p+1)
                            Enfants.append(S)
            if a == b+p:
                if r1 == 0:
                    if q1 <= n-1:
                        if P[q1-1][p-2] == 0 and P[q1][p-1] == 0 and P[q1][p-2] == 0 and
P[q1][p-3] == 0:
                            S = copy.deepcopy(Noeud)
                            S[2][i] = (a+p-2, b+2*p-1)
                            Enfants.append(S)
                elif r1 >= 3:
                    if q1 <= n-2:
                        if P[q1][r1-2] == 0 and P[q1+1][r1-1] == 0 and P[q1+1][r1-2] == 0
and P[q1+1][r1-3] == 0:
                            S = copy.deepcopy(Noeud)
                            S[2][i] = (a+p-2, b+2*p-1)
                            Enfants.append(S)
```

```
    # Tourner avant gauche
    for i in range (len(Emplacements_voitures)):
        a, b = Emplacements_voitures[i]
        q1, r1 = divmod(a, p)
        q2, r2 = divmod(b, p)

        # Voiture horizontale
        if a == b-1:
            if r1 != 1:
                if q1 <= n-3:
                    if P[q1+1][r1-1] == 0 and P[q1][r1-2] == 0 and P[q1+1][r1-2] == 0
and P[q1+2][r1-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-1+2*p, b-2+p)
                        Enfants.append(S)
        if a == b+1:
            if q1 >= 2:
                if 1<= r1 <= p-1:
                    if P[q1-1][r1-1] == 0 and P[q1][r1] == 0 and P[q1-1][r1] == 0 and
P[q1-2][r1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+1-2*p, b+2-p)
                        Enfants.append(S)

        # Voiture verticale
        if a == b-p:
            if r1 == 0:
                if q1 >= 2:
                    if P[q1-1][p-2] == 0 and P[q1-2][p-1] == 0 and P[q1-2][p-2] == 0
and P[q1-2][p-3] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-p-2, b-2*p-1)
                        Enfants.append(S)
            elif r1 >= 3:
                if q1 >= 1:
                    if P[q1][r1-2] == 0 and P[q1-1][r1-1] == 0 and P[q1-1][r1-2] == 0
and P[q1-1][r1-3] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-p-2, b-2*p-1)
                        Enfants.append(S)
        if a == b+p:
            if q1 <= n-2:
                if 1 <= r1 <= p-2:
                    if P[q1][r1] == 0 and P[q1+1][r1-1] == 0 and P[q1+1][r1] == 0 and
P[q1+1][r1+1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+p+2, b+2*p+1)
                        Enfants.append(S)

    # Tourner arrière droit
    for i in range (len(Emplacements_voitures)):
        a, b = Emplacements_voitures[i]
        q1, r1 = divmod(a, p)
        q2, r2 = divmod(b, p)

        # Voiture horizontale
        if a == b-1:
            if q2 >= 2:
                if 1 <= r2 <= p-1:
                    if P[q2-1][r2-1] == 0 and P[q2][r2] == 0 and P[q2-1][r2] == 0 and
P[q2-2][r2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2-p, b+1-2*p)
                        Enfants.append(S)
        if a == b+1:
            if q2 <= n-3:
```



```
                if r2 >= 2:
                    if P[q2+1][r2-1] == 0 and P[q2][r2-2] == 0 and P[q2+1][r2-2] == 0
and P[q2+2][r2-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2+p, b-1+2*p)
                        Enfants.append(S)

        # Voiture verticale
        if a == b-p:
            if q2 <= n-2:
                if 1 <= r2 <= p-2:
                    if P[q2][r2] == 0 and P[q2+1][r2-1] == 0 and P[q2+1][r2] == 0 and
P[q2+1][r2+1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2*p+1, b+p+2)
                        Enfants.append(S)
        if a == b+p:
            if r2 == 0:
                if q2 >= 2:
                    if P[q2-1][p-2] == 0 and P[q2-2][p-1] == 0 and P[q2-2][p-2] == 0
and P[q2-2][p-3] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2*p-1, b-p-2)
                        Enfants.append(S)
            elif r2 >= 3:
                if q2 >= 1:
                    if P[q2][r2-2] == 0 and P[q2-1][r2-1] == 0 and P[q2-1][r2-2] == 0
and P[q2-1][r2-3] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2*p-1, b-p-2)
                        Enfants.append(S)

    # Tourner arrière gauche
    for i in range (len(Emplacements_voitures)):
        a, b = Emplacements_voitures[i]
        q1, r1 = divmod(a, p)
        q2, r2 = divmod(b, p)

        # Voiture horizontale
        if a == b-1:
            if q1 <= n-3:
                if r1 <= p-2:
                    if P[q1+1][r1] == 0 and P[q1][r1+1] == 0 and P[q1+1][r1+1] == 0
and P[q1+2][r1+1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2+p, b+1+2*p)
                        Enfants.append(S)
        if a == b+1:
            if q2 >= 2:
                if r2 >= 2:
                    if P[q2-1][r2-1] == 0 and P[q2][r2-2] == 0 and P[q2-1][r2-2] == 0
and P[q2-2][r2-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2-p, b-1-2*p)
                        Enfants.append(S)

        # Voiture verticale
        if a == b-p:
            if r2 == 0:
                if q2 <= n-1:
                    if P[q2-1][p-2] == 0 and P[q2][p-1] == 0 and P[q2][p-2] == 0 and
P[q2][p-3] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2*p-1, b+p-2)
                        Enfants.append(S)
            elif r2 >= 3:
                if q2 <= n-2:
```

```
                    if P[q2][r2-2] == 0 and P[q2+1][r2-1] == 0 and P[q2+1][r2-2] == 0
and P[q2+1][r2-3] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2*p-1, b+p-2)
                        Enfants.append(S)
        if a == b+p:
            if 1 <= r2 <= p-2:
                if q2 >= 1:
                    if P[q2][r2] == 0 and P[q2-1][r2-1] == 0 and P[q2-1][r2] == 0 and
P[q2-1][r2+1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2*p+1, b-p+2)
                        Enfants.append(S)

    # Avancer avant droit
    for i in range (len(Emplacements_voitures)):
        a, b = Emplacements_voitures[i]
        q1, r1 = divmod(a, p)
        q2, r2 = divmod(b, p)

        # Voiture horizontale
        if a == b-1:
            if q1 >= 1:
                if r1 >= 3:
                    if P[q1][r1-2] == 0 and P[q1-1][r1-1] == 0 and P[q1-1][r1-2] == 0
and P[q1-1][r1-3] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2-p, b-2-p)
                        Enfants.append(S)
        if a == b+1:
            if 1 <= r1 <= p-1:
                if q1 <= n-2:
                    if P[q1][r1] == 0 and P[q1+1][r1-1] == 0 and P[q1+1][r1] == 0 and
P[q1+1][r1+1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2+p, b+2+p)
                        Enfants.append(S)

        # Voiture verticale
        if a == b-p:
            if r1 != 0:
                if q1 >= 2:
                    if P[q1-1][r1-1] == 0 and P[q1][r1] == 0 and P[q1-1][r1] == 0 and
P[q1-2][r1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+1-2*p, b+1-2*p)
                        Enfants.append(S)
        if a == b+p:
            if r1 == 0:
                if q1 <= n-2:
                    if P[q1][p-1] == 0 and P[q1-1][p-2] == 0 and P[q1][p-2] == 0 and
P[q1+1][p-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i]=(a+2*p-1, b+2*p-1)
                        Enfants.append(S)
            elif r1 >= 2:
                if q1 <= n-3:
                    if P[q1+1][r1-1] == 0 and P[q1][r1-2] == 0 and P[q1+1][r1-2] == 0
and P[q1+2][r1-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2*p-1, b+2*p-1)
                        Enfants.append(S)

    # Avancer avant gauche
    for i in range (len(Emplacements_voitures)):
        a, b = Emplacements_voitures[i]
        q1, r1 = divmod(a, p)
```



```
        q2, r2 = divmod(b, p)

        # Voiture horizontale
        if a == b-1:
            if q1 <= n-2:
                if r1 >= 3:
                    if P[q1][r1-2] == 0 and P[q1+1][r1-1] == 0 and P[q1+1][r1-2] == 0
and P[q1+1][r1-3] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2+p, b-2+p)
                        Enfants.append(S)
        if a == b+1:
            if 1 <= r1 <= p-2:
                if q1 >= 1:
                    if P[q1][r1] == 0 and P[q1-1][r1-1] == 0 and P[q1-1][r1] == 0 and
P[q1-1][r1+1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2-p, b+2-p)
                        Enfants.append(S)
        if a == b-p:
            if r1 == 0:
                if q1 >= 3:
                    if P[q1-2][p-1] == 0 and P[q1-1][p-2] == 0 and P[q1-2][p-2] == 0
and P[q1-3][p-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2*p-1, b-2*p-1)
                        Enfants.append(S)
            elif r1 >= 2:
                if q1 >= 2:
                    if P[q1-1][r1-1] == 0 and P[q1][r1-2] == 0 and P[q1-1][r1-2] == 0
and P[q1-2][r1-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2*p-1, b-2*p-1)
                        Enfants.append(S)

        # Voiture verticale
        if a == b+p:
            if 1 <= r1 <= p-1:
                if q1 <= n-3:
                    if P[q1+1][r1-1] == 0 and P[q1][r1] == 0 and P[q1+1][r1] == 0 and
P[q1+2][r1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+1+2*p, b+1+2*p)
                        Enfants.append(S)

    # Reculer arrière droit
    for i in range (len(Emplacements_voitures)):
        a, b = Emplacements_voitures[i]
        q1, r1 = divmod(a, p)
        q2, r2 = divmod(b, p)

        # Voiture horizontale
        if a == b-1:
            if r1 <= p-3:
                if q1 >= 1:
                    if P[q1][r1+1] == 0 and P[q1-1][r1] == 0 and P[q1-1][r1+1] == 0
and P[q1-1][r1+2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2-p, b+2-p)
                        Enfants.append(S)
        if a == b+1:
            if r2 >= 3:
                if q2 <= n-2:
                    if P[q2][r2-2] == 0 and P[q2+1][r2-1] == 0 and P[q2+1][r2-1] == 0
and P[q2+1][r2-1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2+p, b-2+p)
```

```
                        Enfants.append(S)

        # Voiture verticale
        if a == b-p:
            if 1 <= r1 <= p-1:
                if q1 <= n-4:
                    if P[q1+2][r1-1] == 0 and P[q1+1][r1] == 0 and P[q1+2][r1] == 0
and P[q1+3][r1] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2*p+1, b+2*p+1)
                        Enfants.append(S)
        if a == b+p:
            if r2 == 0:
                if q2 >= 3:
                    if P[q2-2][p-1] == 0 and P[q2-1][p-2] == 0 and P[q2-2][p-2] == 0
and P[q2-3][p-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2*p-1, b-2*p-1)
                        Enfants.append(S)
            elif r2 >= 2:
                if q2 >= 2:
                    if P[q2-1][r2-1] == 0 and P[q2][r2-2] == 0 and P[q2-1][r2-2] == 0
and P[q2-2][r2-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2*p-1, b-2*p-1)
                        Enfants.append(S)

    # Reculer arrière gauche
    for i in range (len(Emplacements_voitures)):
        a, b = Emplacements_voitures[i]
        q1, r1 = divmod(a, p)
        q2, r2 = divmod(b, p)

        # Voiture horizontale
        if a == b-1:
            if q1 <= n-2:
                if r1 <= p-3:
                    if P[q1][r1+1] == 0 and P[q1+1][r1] == 0 and P[q1+1][r1+1] == 0
and P[q1+1][r1+2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2+p, b+2+p)
                        Enfants.append(S)
        if a == b+1:
            if r2 >= 3:
                if q2 >= 1:
                    if P[q2][r2-2] == 0 and P[q2-1][r2-1] == 0 and P[q2-1][r2-2] == 0
and P[q2-1][r2-3] ==0 :
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2-p, b-2-p)
                        Enfants.append(S)

        # Voiture verticale
        if a == b-p:
            if r2 == 0:
                if q2 <= n-2:
                    if P[q2][p-1] == 0 and P[q2-1][p-2] == 0 and P[q2][p-2] == 0 and
P[q2+1][p-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2*p-1, b+2*p-1)
                        Enfants.append(S)
            elif r2 >= 2:
                if q2 <= n-3:
                    if P[q2+1][r2-1] == 0 and P[q2][r2-2] == 0 and P[q2+1][r2-2] == 0
and P[q2+2][r2-2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a+2*p-1, b+2*p-1)
                        Enfants.append(S)
```



```
        if a == b+p:
            if r2 != 0:
                if q2 >= 2:
                    if P[q2-1][r2-1] == 0 and P[q2][r2] == 0 and P[q2-1][r2] == 0 and
P[q2-2][r2] == 0:
                        S = copy.deepcopy(Noeud)
                        S[2][i] = (a-2*p+1, b-2*p+1)
                        Enfants.append(S)

    return Enfants
```

```
def heuristique(Noeud_de_depart, indice_voiture_cible):

    # Initialisation des données
    dimensions, k, Emplacements_voitures = Noeud_de_depart
    n, p = dimensions
    a, b = Emplacements_voitures[indice_voiture_cible]
    q1, r1 = divmod(a, p)
    q2, r2 = divmod(b, p)

    # Détermination du plus court chemin

    # Voiture horizontale
    if a == b-1:
        if q1 < n-2:
            return abs(r1-2)+4+abs(n-(q1+1+2))
        else:
            return abs(r1-2)+4+abs(n-1-(q1+1-2))
    if a == b+1:
        if q2 < n-2:
            return abs(r2-2)+4+abs(n-(q2+1+2))
        else:
            return abs(r2-2)+4+abs(n-1-(q2+1-2))

    # Voiture verticale
    if a == b+p:
        if r1 == 1:
            return abs(n-(q1+1))
        elif r1 == 2:
            if q1 <= n-2:
                return 4+abs(n-(q1+1+2))
            elif q1 > n-2:
                return 4+abs(n-(q1+1-2))
        elif r1 == 3:
            if q1 <= n-3:
                return 4+abs(n-2-(q1+1+2))+4
            else:
                return 4+abs(n-2-(q1+1-2))+4
        elif r1 != 0:
            if q1 <= n-3:
                return 4+abs(2-(r1-2))+4+abs(n-(q1+1+3))
            else:
                return 4+abs(2-(r1-2))+4+abs(n-(q1+1))
        else:
            if n == 4:
                return 4+4+abs(n-q1)
            else:
                if q1-1 <= n-3:
                    return 4+abs(2-(p-2))+4+abs(n-(q1+3))
                else:
                    return 4+abs(2-(p-2))+4+abs(n-q1)
    if a == b-p:
        if r2 == 1:
            return abs((n-1)-q2)
        elif r2 == 2:
```

```
            if q2 <= n-2:
                return 4+abs(n-(q2+1+2))
            elif q2 > n-2:
                return 4+abs(n-(q2+1-2))
        elif r2 == 3:
            if q2 <= n-3:
                return 4+abs(n-2-(q2+1+2))+4
            else:
                return 4+abs(n-2-(q2+1-2))+4
        elif r2 != 0:
            if q2 <= n-3:
                return 4+abs(2-(r2-2))+4+abs(n-(q2+1+3))
            else:
                return 4+abs(2-(r2-2))+4+abs(n-(q2+1))
        else:
            if n == 4:
                return 4+4+abs(n-q2)
            else:
                if q2-1 <= n-3:
                    return 4+abs(2-(p-2))+4+abs(n-(q2+3))
                else:
                    return 4+abs(2-(p-2))+4+abs(n-q2)
```

```
def cout(Noeud_1, Noeud_2):

    # Initialisation des données
    dimensions, k, Emplacements_voitures_1 = Noeud_1
    dimensions, k, Emplacements_voitures_2 = Noeud_2
    n, p = dimensions

    # Indice de la voiture qui s'est déplacée
    i0 = 0
    for i in range (len(Emplacements_voitures_1)):
        if Emplacements_voitures_1[i] != Emplacements_voitures_2[i]:
            i0 = i

    # Position de la voiture dans chaque parking
    a1, b1 = Emplacements_voitures_1[i0]
    a2, b2 = Emplacements_voitures_2[i0]

    # Cas où la voiture ne s'est pas déplacée
    if a2 == a1 and b2 == b1:
        return 0

    # Autres cas
    if a1 == b1-1:
        if a2 == a1-1 and b2 == b1-1:
            return 1
        if a2 == a1+1 and b2 ==b1+1:
            return 1
        else:
            return 4

    if a1 == b1+1:
        if a2 == a1+1 and b2 == b1+1:
            return 1
        if a2 == a1-1 and b2 == b1-1:
            return 1
        else:
            return 4

    if a1 == b1-p:
        if a2 == a1-p and b2 == b1-p:
            return 1
        if a2 == a1+p and b2 == b1+p:
```



```
            return 1
        else:
            return 4

    if a1 == b1+p:
        if a2 == a1+p and b2 == b1+p:
            return 1
        if a2 == a1-p and b2 == b1-p:
            return 1
        else:
            return 4
```

```
def cout_actuel(Noeud_de_depart, Noeud_actuel, Relation_pere_fils):

    # Initialisation des données
    c = 0
    Noeud = Noeud_actuel

    # Calcul du poids
    while Noeud != Noeud_de_depart:
        for elt in Relation_pere_fils:
            if elt[1] == Noeud :
                Noeud = elt[0]
                c += elt[2]
    c += cout(Noeud_de_depart, Noeud)

    return c
```

```
def algorithme_A_etoile(Noeud_de_depart, Voiture_cible):

    # Initialisation des données
    dimensions, k, Emplacements_voitures = Noeud_de_depart
    n, p = dimensions
    a, b = Voiture_cible

    # Indice permettant de retrouver la position de la voiture cible au sein des
différents noeuds parcourus
    indice_voiture_cible = 0
    for i in range (len(Emplacements_voitures)):
        if Emplacements_voitures[i] == Voiture_cible:
            indice_voiture_cible = i

    # Liste contenant les noeuds à visiter ainsi que leur valeur de f correspondante,
classés selon leur valeur par f
    Noeuds_a_visiter_poids = [(Noeud_de_depart, 0 + heuristique(Noeud_de_depart,
indice_voiture_cible))]

    # Liste contenant les noeuds à visiter, classés selon leur valeur par la fonction
f où f(Noeud) = heurisitique(Noeud) + cout_actuel(Noeud)
    Noeuds_a_visiter = [Noeud_de_depart]

    # Liste contenant au fur et à mesure les noeuds déjà visités
    Noeuds_visites = []

    # Liste contenant les relations "père fils" entre les différents noeuds ainsi que
le coût pour passer entre ces deux noeuds
    Relation_pere_fils=[]
    while Noeuds_a_visiter != []:

        # On choisit le noeud le plus prometteur
        Noeud_actuel = Noeuds_a_visiter_poids[0][0]

        if Noeud_actuel[2][indice_voiture_cible] == (n*p-(p-1), n*p-(p-1)-p) or
```

```
Noeud_actuel[2][indice_voiture_cible] == (n*p-(p-1)-p, n*p-(p-1)): # La voiture cible
est située à la sortie
            return Noeuds_a_visiter_poids[0][1]

        else:
            Noeuds_visites.append(Noeud_actuel)

            # On génère les fils du noeud actuel
            Enfants = enfants_noeud(Noeud_actuel)
            for Fils in Enfants:
                if Fils not in Noeuds_a_visiter and Fils not in Noeuds_visites:
                    Relation_pere_fils.append((Noeud_actuel, Fils, cout(Noeud_actuel,
Fils)))
                    f = cout_actuel(Noeud_de_depart, Fils, Relation_pere_fils) +
heuristique(Fils, indice_voiture_cible)

                    # On place les différents fils du noeud actuel dans la liste des
noeuds à visiter, en fonction de leur valeur par f
                    m = 0
                    while m < len(Noeuds_a_visiter_poids) and f >=
cout_actuel(Noeud_de_depart, Noeuds_a_visiter_poids[m][0], Relation_pere_fils) +
heuristique(Noeuds_a_visiter_poids[m][0], indice_voiture_cible):
                        m += 1
                    Noeuds_a_visiter_poids.insert(m, (Fils,
cout_actuel(Noeud_de_depart, Fils, Relation_pere_fils) + heuristique(Fils,
indice_voiture_cible)))
                    Noeuds_a_visiter.insert(m, Fils)

        # On retire le noeud actuel de la liste des noeuds à visiter
        Noeuds_a_visiter_poids.remove((Noeud_actuel, cout_actuel(Noeud_de_depart,
Noeud_actuel, Relation_pere_fils) + heuristique(Noeud_actuel, indice_voiture_cible)))
        Noeuds_a_visiter.remove(Noeud_actuel)

    return 10**3
```

```
def cout_moyen(Noeud):

    # Initialisation des données
    dimensions, k, Emplacements_voitures = Noeud
    n, p = dimensions
    c = 0
    c_moy = 0

    # Cas où le nombre de voitures annoncé ne correspond pas au nombre de voitures
présentes dans le parking
    if k != len(Emplacements_voitures):
        print ('Erreur : le nombre de voitures annoncé ne correspond pas aux nombres
de voitures présentes dans le parking.')
        return None

    # Calcul du coût total
    for Voiture in Emplacements_voitures:
        c += algorithme_A_etoile(Noeud, Voiture)

    # Calcul du coût moyen
    c_moy = c / k

    return c_moy
```

```
def generer_parking_aleatoire(n, p, k):

    # Initialisation des données
    Emplacements_voitures = []
```



```
    # Génération d'emplacements aléatoires
    while len(Emplacements_voitures) != k:

        a = random.randint(1, n*p)

        if a//p == 0:
            if a%p == 1:
                b = random.choice((a+1, a+p))
            elif a%p == 0:
                b = random.choice((a-1, a+p))
            else:
                b = random.choice((a+1, a-1, a+p))

        elif a//p == p-1:
            if a%p == 1:
                b = random.choice((a+1, a-p))
            elif a%p == 0:
                b = random.choice((a-1, a-p))
            else:
                b = random.choice((a+1, a-1, a-p))

        elif a%p == 1:
            if a//p == 0:
                b = random.choice((a+1, a+p))
            elif a//p == p-1:
                b = random.choice((a+1, a-p))
            else:
                b = random.choice((a+1, a+p, a-p))

        elif a%p == 0:
            if a//p == 1:
                b = random.choice((a-1, a+p))
            elif a//p == p:
                b = random.choice((a-1, a-p))
            else:
                b = random.choice((a-1, a+p, a-p))

        else:
            b = random.choice((a+1, a-1, a+p, a-p))

        # Vérification de colisions
        test = True
        for voiture in Emplacements_voitures:
            if a == voiture[0] or a == voiture[1] or b == voiture[0] or b ==
voiture[1]:
                test = False
        if test == True:
            Emplacements_voitures.append((a, b))

    return [(n, p), k, Emplacements_voitures]
```

```
def generer_voisin_aleatoire(Noeud):

    Enfants = enfants_noeud(Noeud)
    voisin_aleatoire = random.choice(Enfants)

    return voisin_aleatoire
```

```
def recuit_simule(n, p, k):

    # Initialisation des données
    temperature_initiale = 0
```

```
    cout_initial = 0
    Noeud_initial = generer_parking_aleatoire(n, p, k)
    cout_initial = cout_moyen(Noeud_initial)

    # Cas d'un parking où aucune voiture ne peut se déplacer
    while cout_initial == 10**3:
        Noeud_initial = generer_parking_aleatoire(n, p, k)
        cout_initial = cout_moyen(Noeud_initial)

    # Génération d'une température initiale
    for i in range (3):
        delta_cout = abs((cout_initial -
cout_moyen(random.choice(enfants_noeud(Noeud_initial)))))
        if delta_cout > temperature_initiale:
            temperature_initiale = delta_cout
    temperature_initiale = temperature_initiale*10

    # Initialisation de la boucle
    temperature = temperature_initiale
    compteur_stagnation = 0
    Noeud_actuel = Noeud_initial

    # Application de l'algorithme
    while temperature > temperature_initiale*(0.9)**40:

        for i in range (10):

            # Stagnation
            if compteur_stagnation == 100:
                break

            # Génération d'un voisin
            Voisin = generer_voisin_aleatoire(Noeud_actuel)

            # Évaluation des solutions
            cout_actuel = cout_moyen(Noeud_actuel)
            cout_voisin = cout_moyen(Voisin)
            delta_cout = cout_voisin - cout_actuel
            if delta_cout < 0:
                Noeud_actuel = Voisin
                compteur_stagnation = 0
            elif delta_cout == 0:
                compteur_stagnation += 1
            else:
                if np.exp(-delta_cout/temperature) > random.uniform(0,1):
                    Noeud_actuel = Voisin
                    compteur_stagnation = 0
                else:
                    compteur_stagnation += 1

        # Refroidissement
        temperature = 0.9*temperature

    return creer_parking(Noeud_actuel), cout_actuel
```