L'utilisation de drones pour la lutte contre le braconnage

Présenté par:

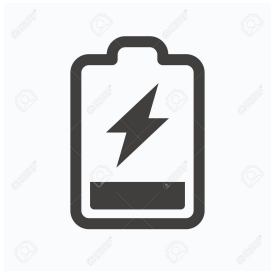
- OKON N'Guessan Paul-Camille N°34113

Partenaires:

- Nina ABAD N°16696
- Elisa ROUSSEAU N°44603

Comment utiliser efficacement des drones pour protéger la faune et la flore des braconniers ?

Sommaire :



- ❖ Déplacement
- ● Alimentation
- ● Détection

Déplacement

Problématique: Comment faire une optimisation du chemin suivi par le drone?

Objectifs:

1. **Mettre le problème en évidence**
2. **Trouver des algorithmes qui proposent des solutions**
3. **Comparer les algorithmes trouvées**

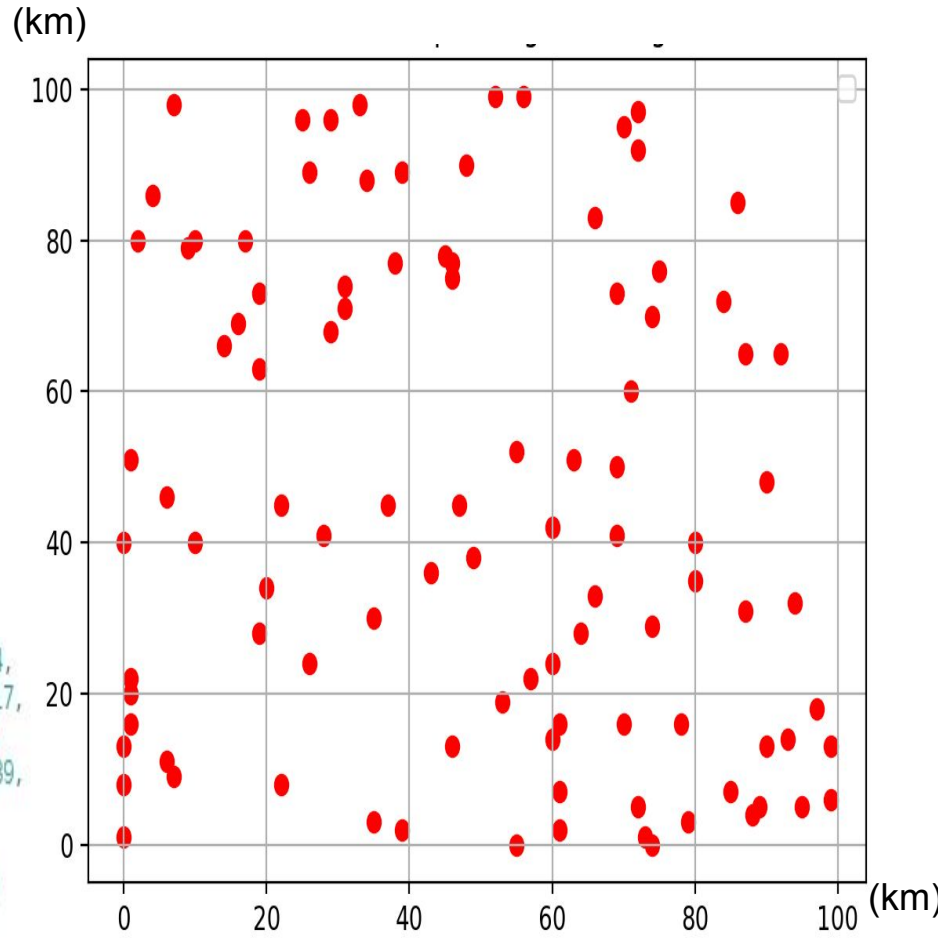
MISE EN ÉVIDENCE DU PROBLÈME

1. *<u>Parcourir aléatoirement des points donnés n'est pas judicieux pour un drone</u>*

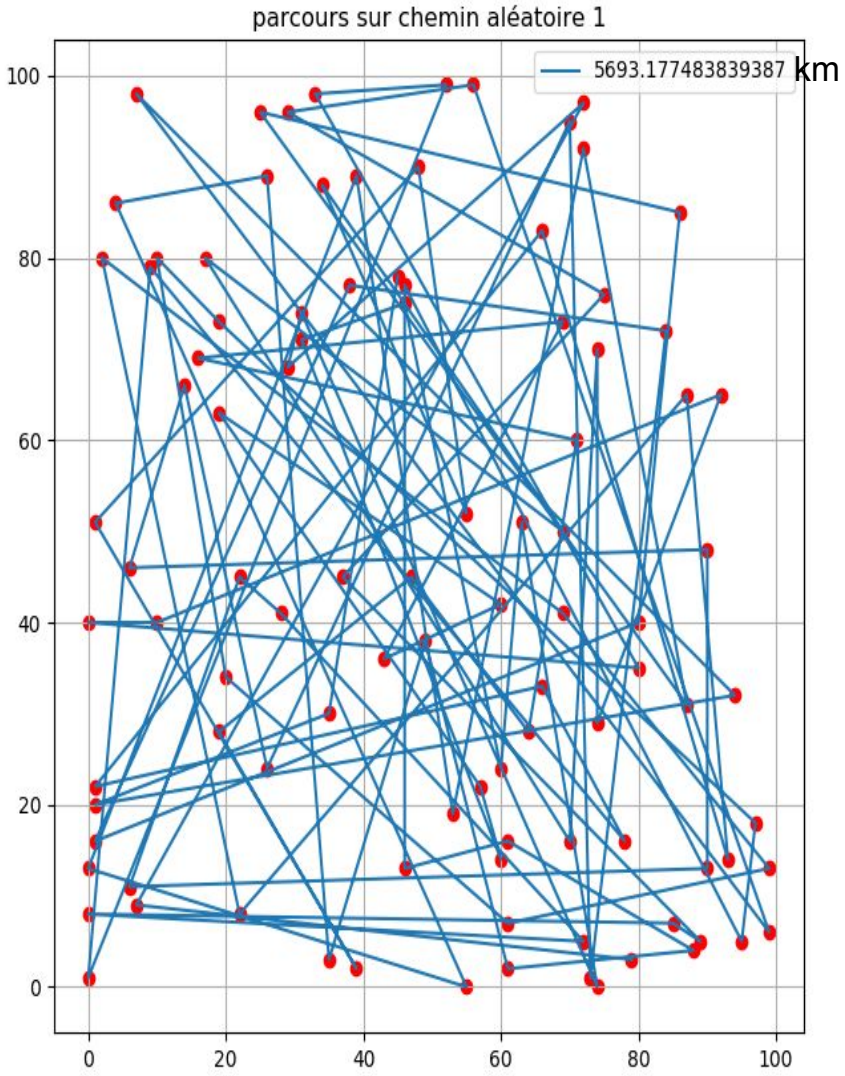
```
pts = [[93, 14], [26, 89], [52, 99], [48, 90], [6, 11], [39, 89], [0, 40], [86, 85],
[29, 68], [79, 3], [97, 18], [53, 19], [94, 32], [0, 1], [89, 5], [69, 41], [19, 73],
[49, 38], [10, 80], [87, 31], [35, 3], [19, 28], [60, 14], [60, 42], [0, 13], [95, 5],
[7, 9], [74, 0], [35, 30], [10, 40], [84, 72], [9, 79], [45, 78], [70, 95], [17, 80],
[26, 24], [92, 65], [16, 69], [1, 16], [69, 73], [38, 77], [1, 51], [80, 35], [72, 92],
[4, 86], [87, 65], [75, 76], [46, 13], [61, 7], [22, 8], [72, 5], [61, 2], [66, 83],
[90, 13], [46, 75], [85, 7], [34, 88], [74, 29], [61, 16], [33, 98], [99, 6], [43, 36],
[56, 99], [72, 97], [70, 16], [90, 48], [55, 0], [37, 45], [20, 34], [55, 52], [14, 66],
[22, 45], [63, 51], [47, 45], [28, 41], [71, 60], [80, 40], [1, 22], [7, 98], [73, 1],
[6, 46], [60, 24], [88, 4], [0, 8], [25, 96], [64, 28], [31, 74], [31, 71], [57, 22],
[66, 33], [74, 70], [19, 63], [46, 77], [2, 80], [69, 50], [78, 16], [1, 20], [99, 13],
[29, 96], [39, 2]]
```

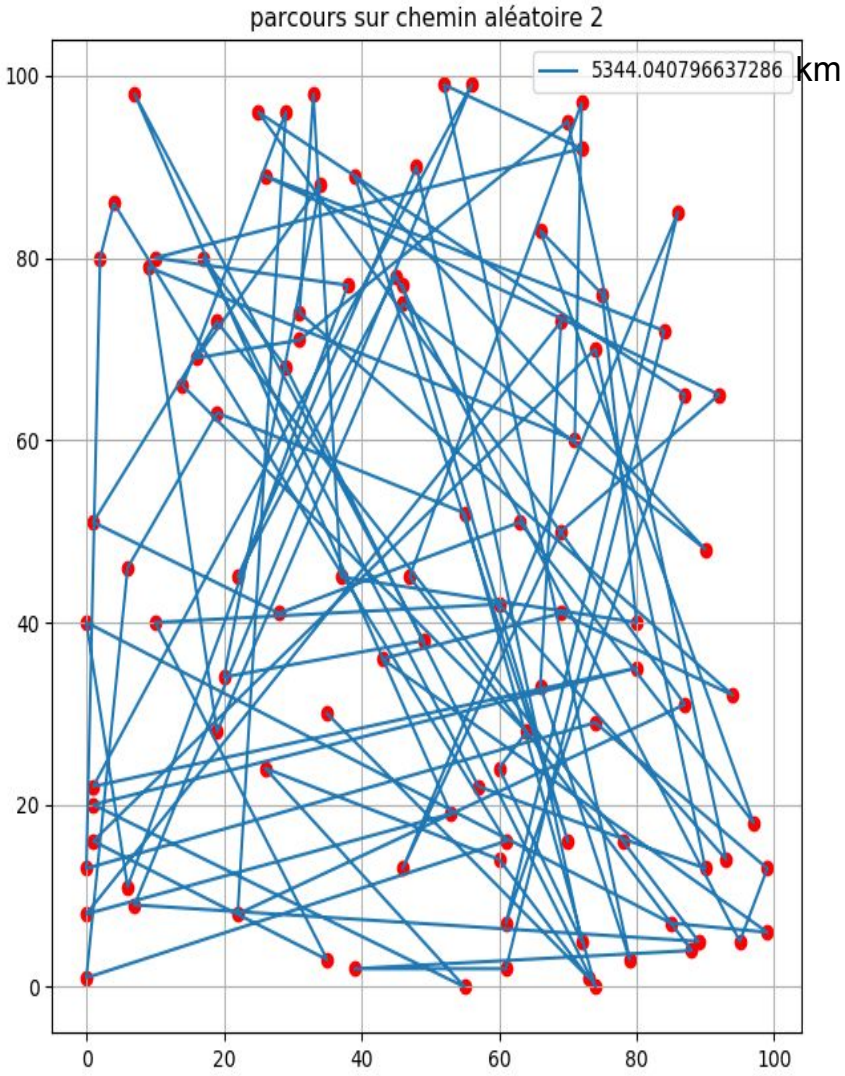
```
chemin_aléatoire1 = [32, 28, 96, 12, 78, 60, 62, 98, 46, 67, 14, 55, 83, 50, 74, 71, 4,
53, 65, 80, 70, 35, 33, 79, 30, 40, 38, 76, 7, 84, 19, 43, 11, 5, 24, 66, 44, 1, 20, 17,
51, 82, 58, 47, 92, 25, 10, 16, 18, 68, 48, 97, 15, 93, 49, 45, 0, 52, 77, 89, 95, 31,
13, 2, 59, 42, 6, 29, 36, 57, 90, 64, 34, 94, 56, 69, 3, 41, 99, 21, 73, 85, 75, 37, 39,
61, 23, 91, 88, 22, 87, 54, 81, 72, 27, 86, 8, 63, 26, 9]

chemin_aléatoire2 = [92, 64, 5, 65, 54, 21, 31, 75, 63, 73, 82, 99, 51, 30, 1, 36, 94,
47, 7, 48, 45, 84, 19, 49, 98, 70, 85, 81, 50, 3, 71, 32, 25, 97, 57, 24, 93, 44, 58,
13, 80, 91, 69, 27, 8, 34, 88, 53, 90, 38, 20, 29, 23, 0, 33, 87, 37, 56, 68, 17, 78,
11, 83, 39, 89, 9, 2, 43, 18, 40, 4, 6, 55, 60, 61, 15, 12, 86, 59, 67, 76, 46, 52, 10,
72, 74, 41, 16, 95, 14, 26, 62, 77, 42, 96, 66, 35, 22, 79, 28]
```



MISE EN ÉVIDENCE DU PROBLÈME





MISE EN ÉVIDENCE DU PROBLÈME

2. *Lister tous les chemins possibles à un temps de calcul trop important pour un ordinateur.*

Nombre de chemins possible: n!
Pour 20 points: 2,4 10^8 chemins soit 8.10^4 années de calcul
La longueur de ce calcul le rendrait impossible en pratique.
L'algorithme n'est pas exploitable

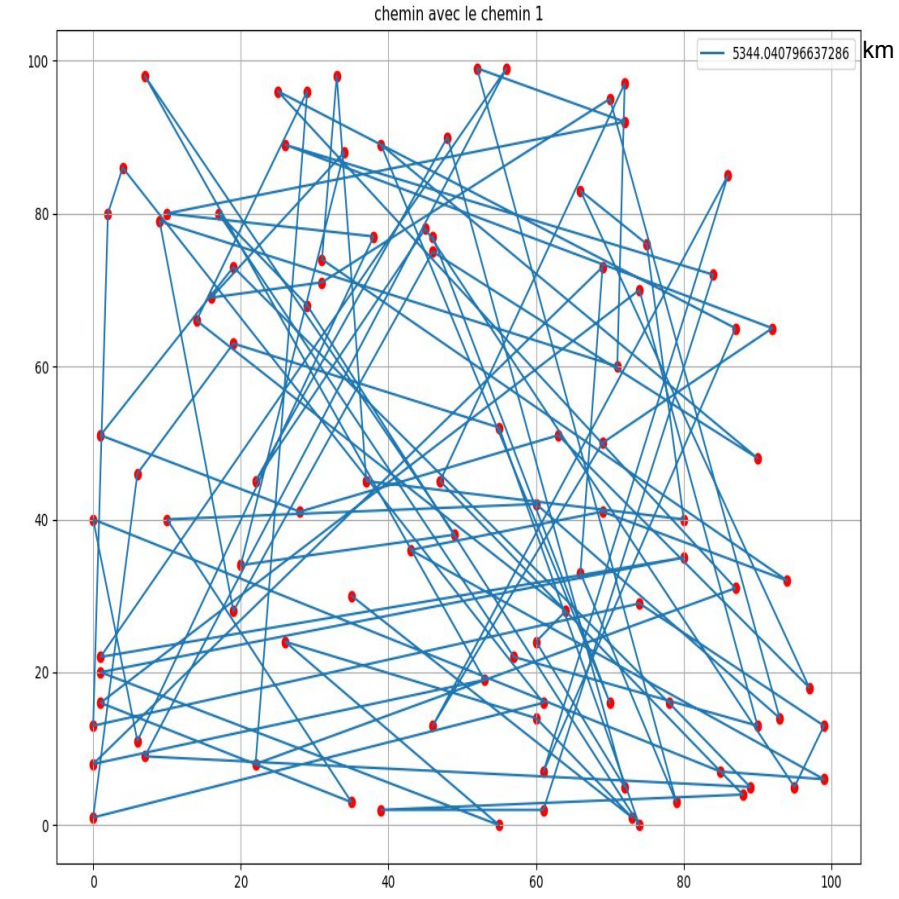
RECHERCHE DE SOLUTIONS OPTIMALES

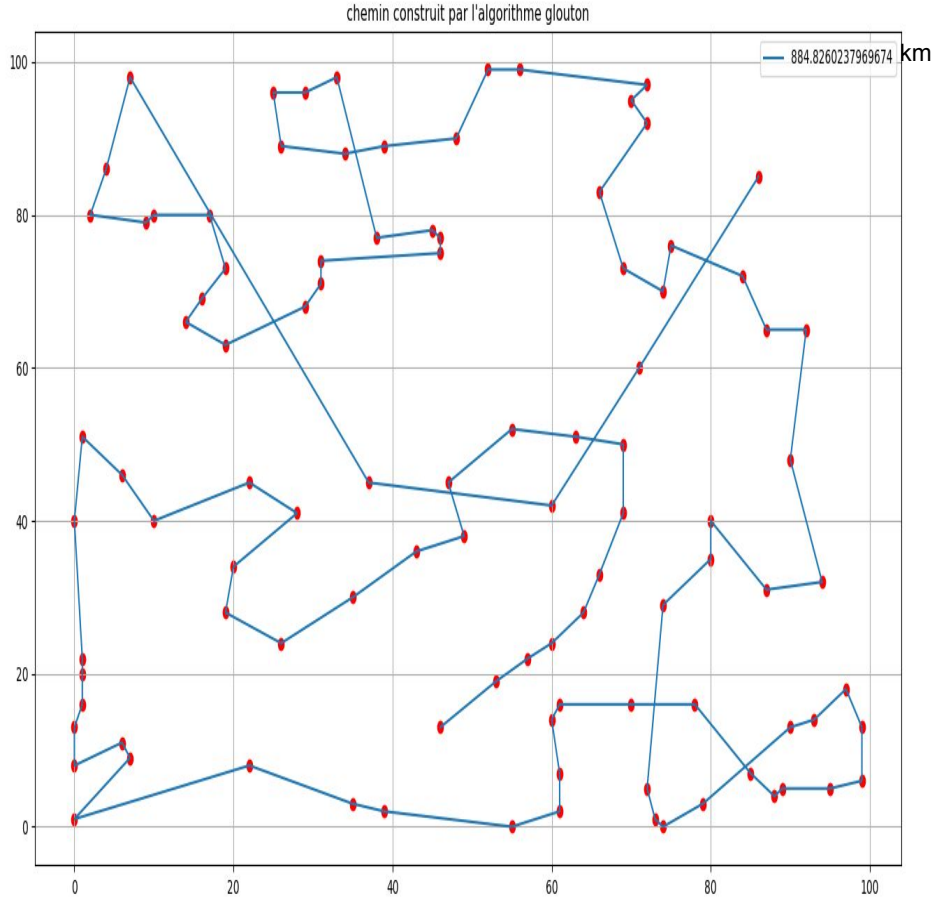
1. *Algorithme Glouton: Méthode du plus proche voisin*

Principe: Parcourir les points en choisissant le plus proche à chaque étape.

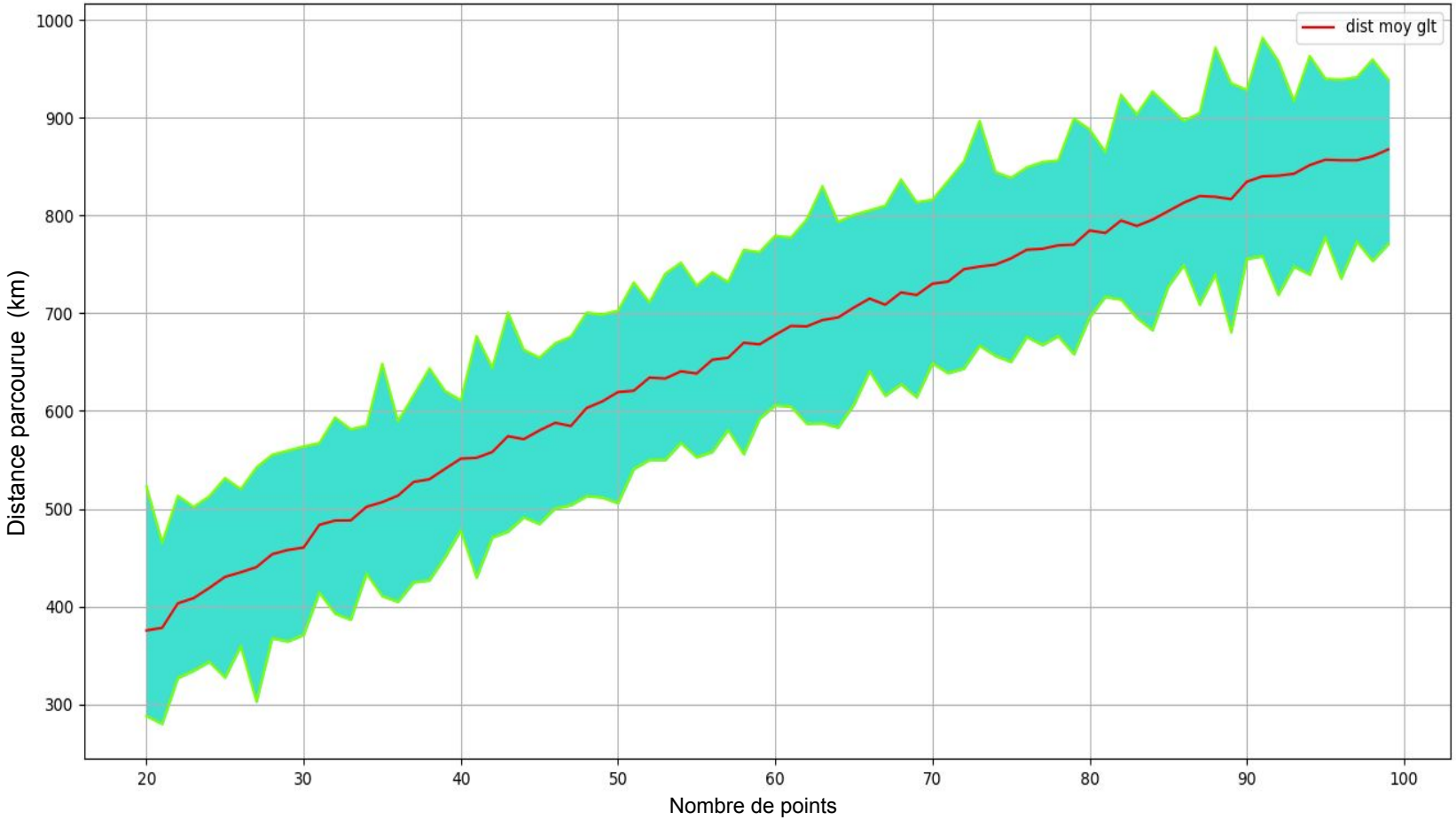
Complexité: $O(n^3)$

RECHERCHE DE SOLUTIONS OPTIMALES



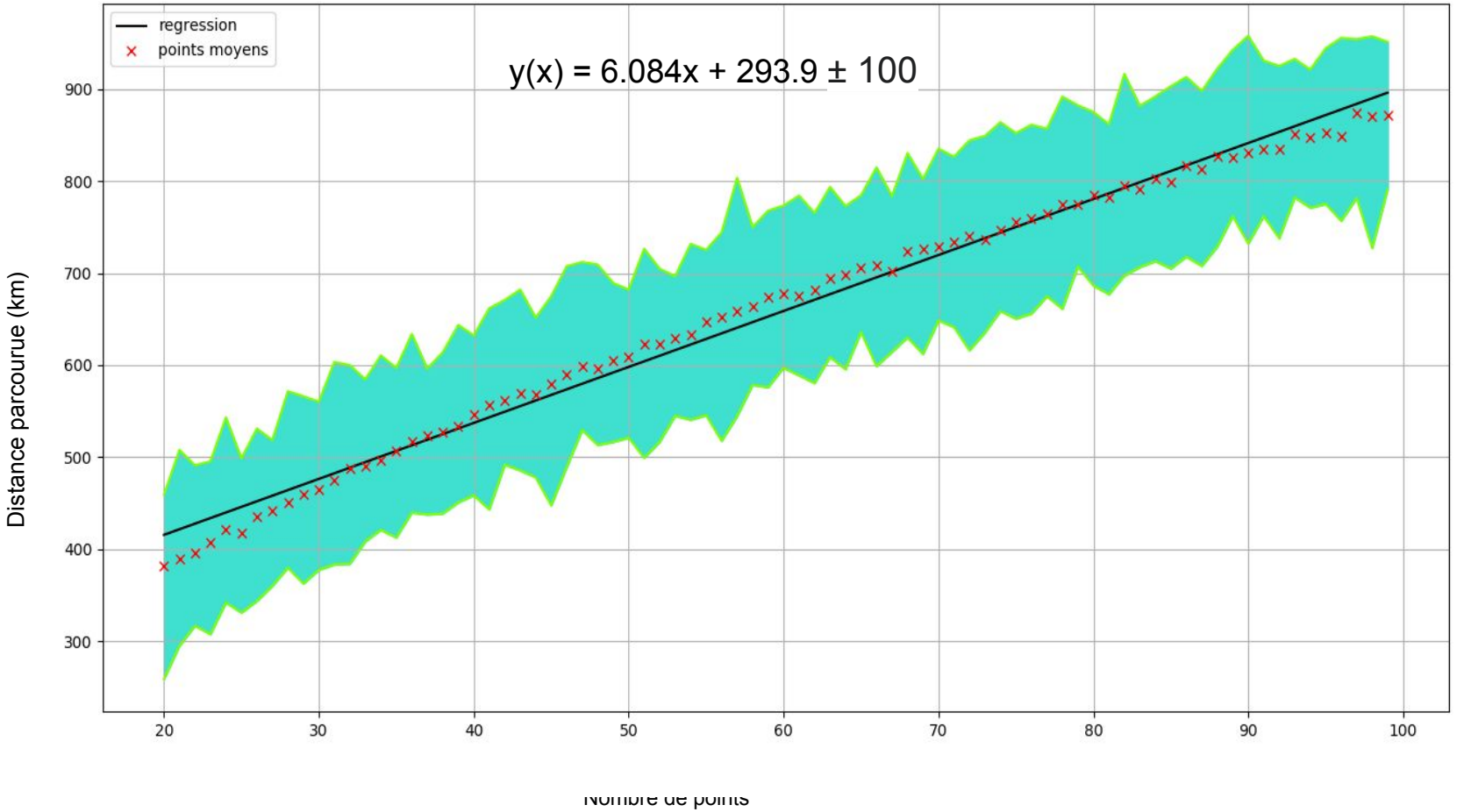


RECHERCHE DE SOLUTIONS OPTIMALES



Distance parcourue en fonction du nombre de points

RECHERCHE DE SOLUTIONS OPTIMALES



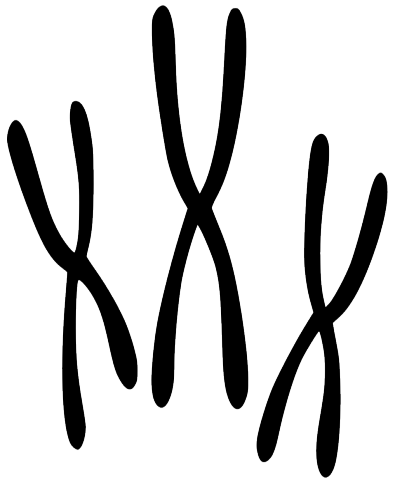
Distance parcourue en fonction du nombre de points

RECHERCHE DE SOLUTIONS OPTIMALES

2. *Algorithme génétique*

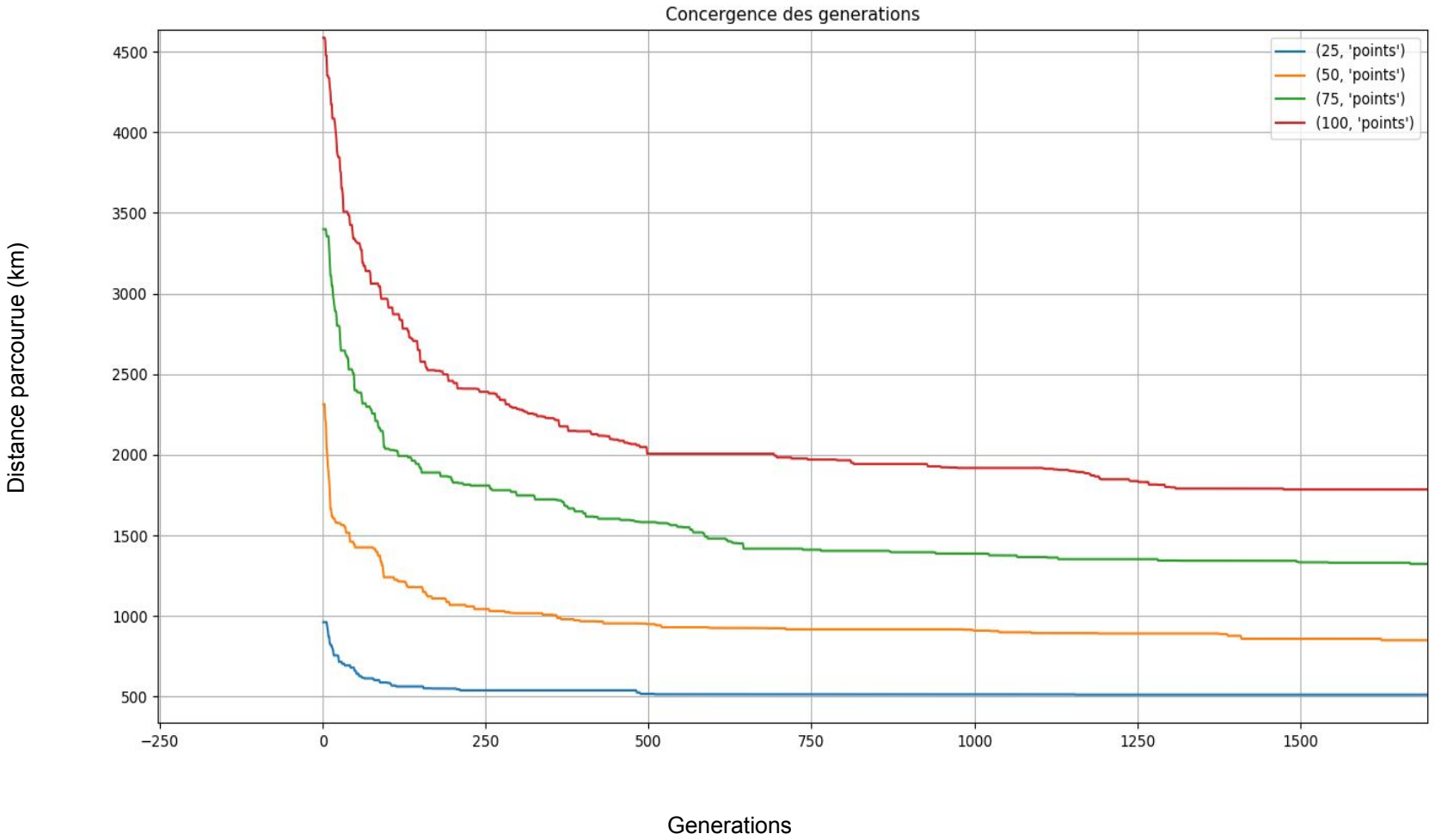
Principe: Les algorithmes génétiques s'inspirent de la théorie de l'évolution en simulant l'évolution d'une population. Ils font intervenir trois grands traitements.
On considère n points

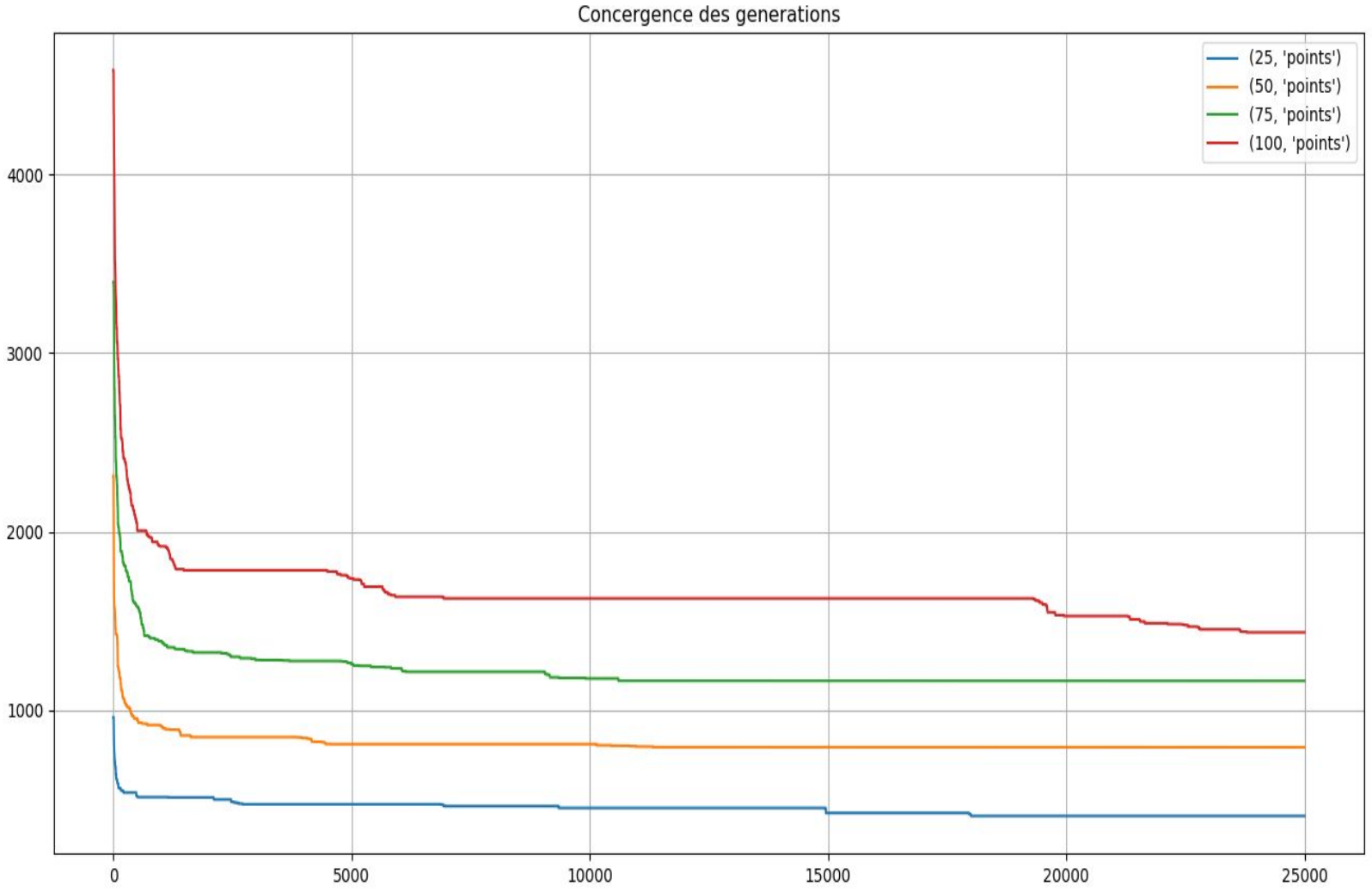
1. Initialisation et Évaluation: m individus, O(m.n)
2. Sélection: O(m.log(m))
3. Mutation et Croisement: O(m.n)



Complexité générale: Pour une exécution sur g générations: $O(n^2+m.g.log(m)+m.g.n)$

RECHERCHE DE SOLUTIONS OPTIMALES




Concergence des generations
(25, 'points')
(50, 'points')
(75, 'points')
(100, 'points')
4000
3000
2000
1000
0
5000
10000
15000
20000
25000

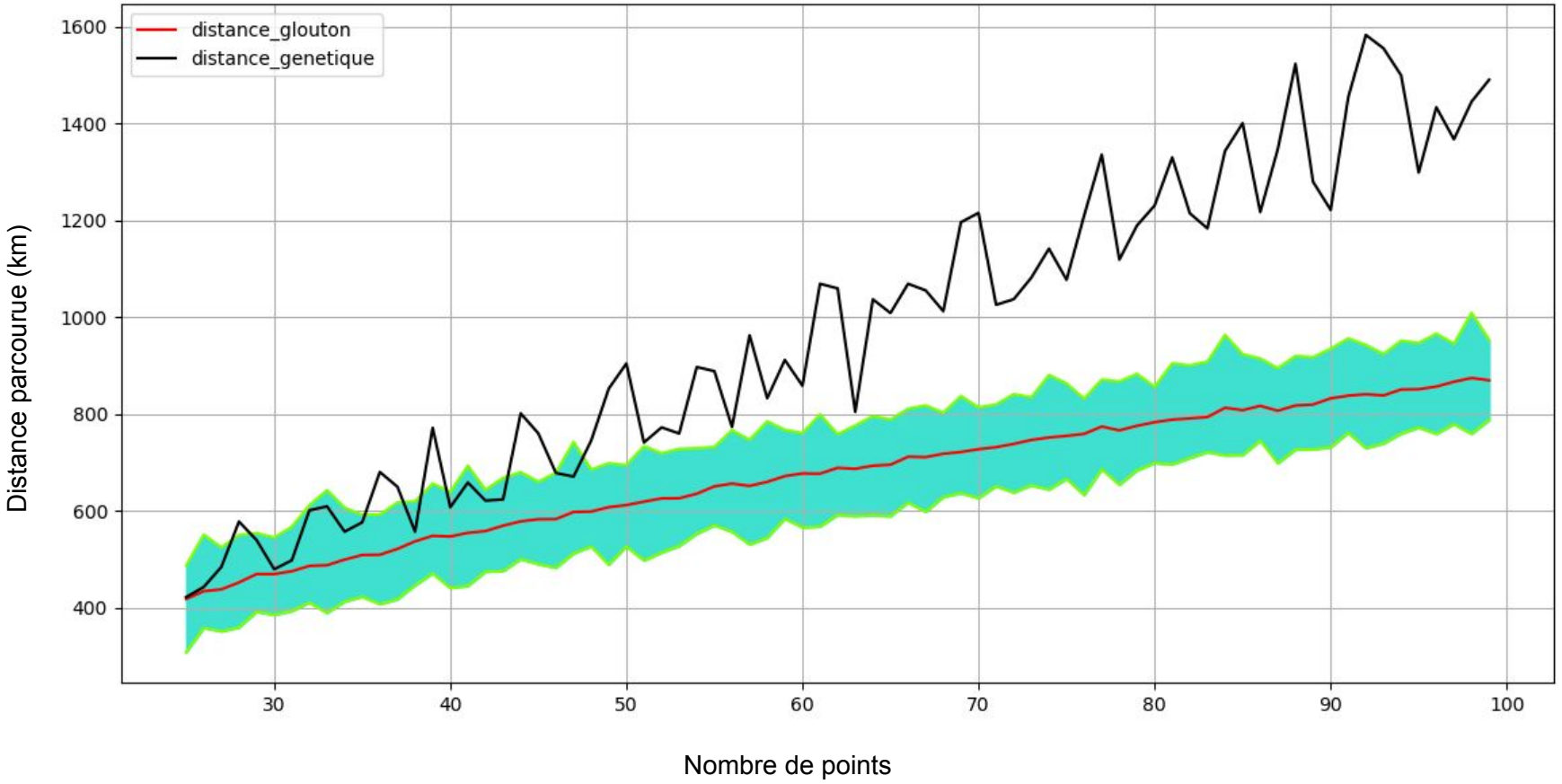
COMPARER LES ALGORITHMES POUR OPTIMISER LES CHOIX

Algorithme glouton: $O(n^3)$

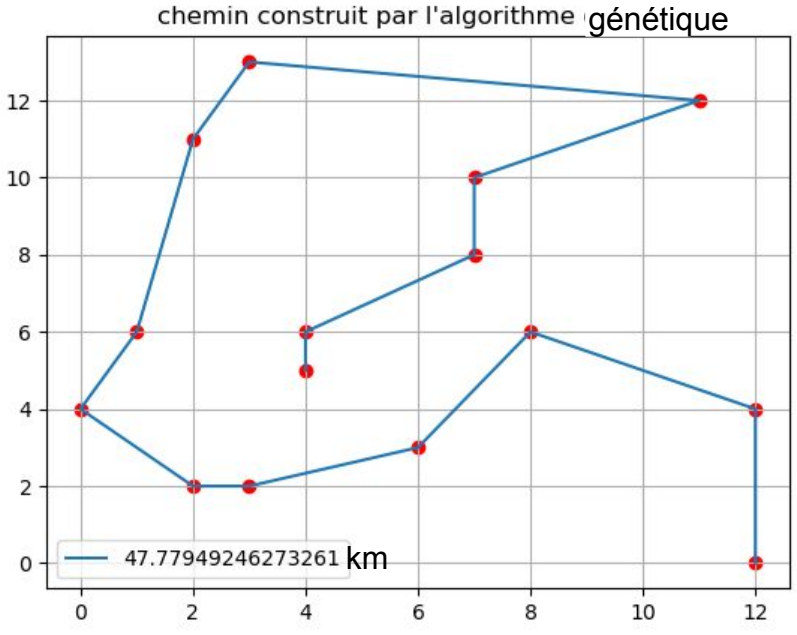
Algorithme génétique: $O(n^2+m.log(m)+m.g.n))$

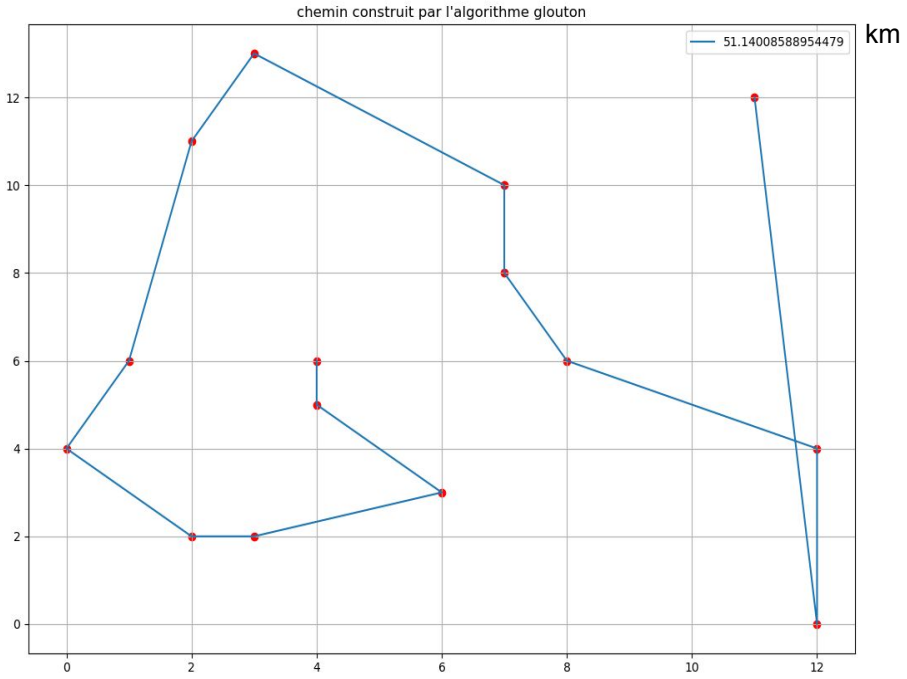
COMPARER LES ALGORITHMES POUR OPTIMISER LES CHOIX

g = 10000 ;
m = 300

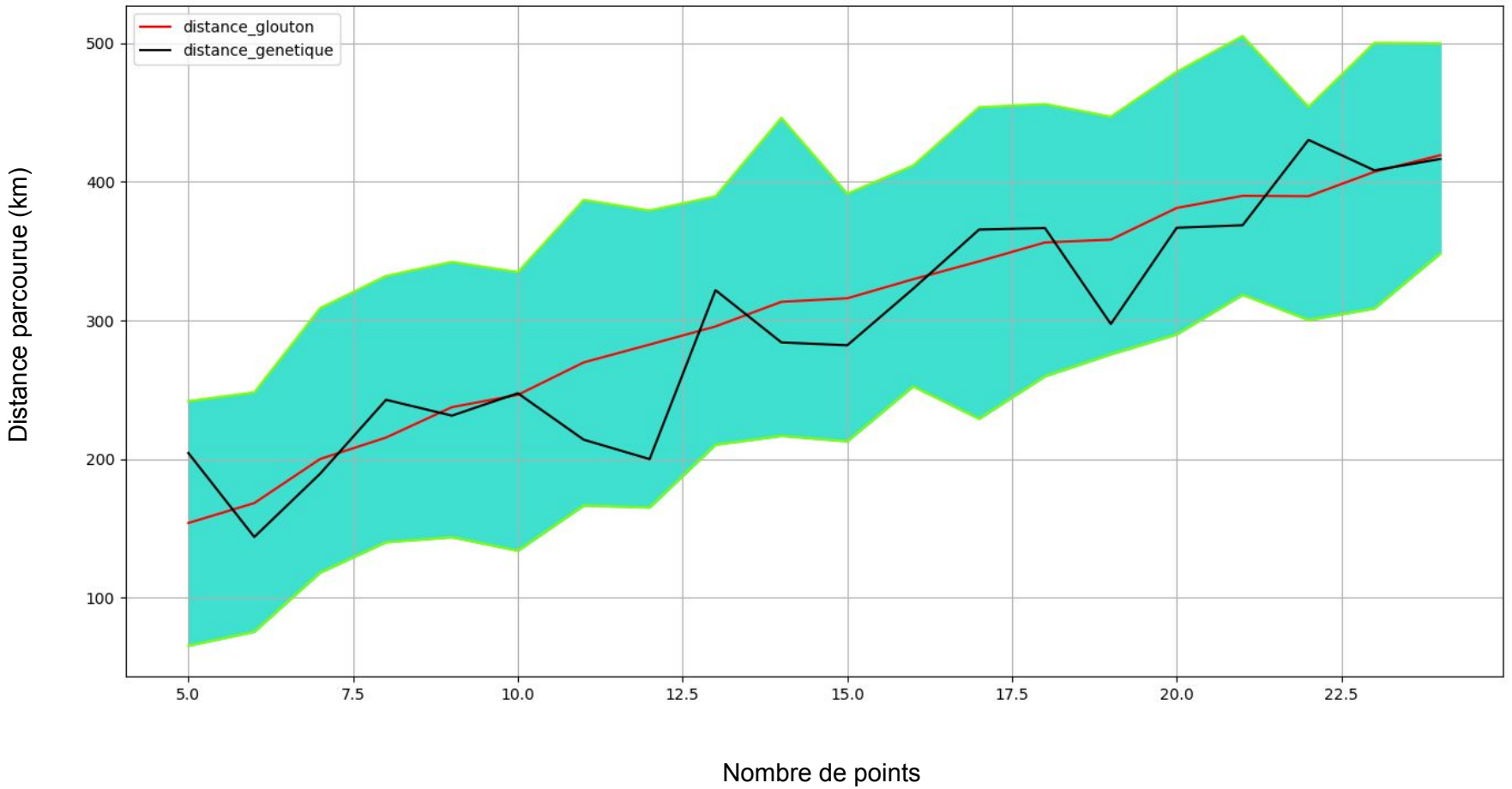


COMPARER LES ALGORITHMES POUR OPTIMISER LES CHOIX





COMPARER LES ALGORITHMES POUR OPTIMISER LES CHOIX



CONCLUSION

- Algorithme génétique: efficace pour peu de points
- Algorithme glouton: plus efficace pour beaucoup de points

ANNEXE

https://www.drone-malin.com/blog/acheter-un-drone-les-bonnes-adresses-de-drone-malin.html

https://www.yvelines.fr/jeunesse/education/orientation/

https://fr.123rf.com/photo_60866584_batterie-ic%C3%B4ne-d-alimentation-charge-de-symbole-de-l-accumulateur-gris-ic%C3%B4ne-web-plat-sur-fond-blanc.html

https://www.bosch-professional.com/fr/fr/instruments-de-mesure/gtc-400c.html

ANNEXE

```python
import numpy as np
import random as rd
import random
import matplotlib.pyplot as plt
```

```python
## FONCTIONS UTILES

distance = lambda A,B : np.sqrt((A[0]-B[0])**2 + (A[1]-B[1])**2)

def generer_pts(n,coord_max):                                   # Pour generer aleatoire un ensemble de point par lesquels le drone doit passer
    pts = []
    while len(pts) < n:
        x = rd.randrange(0,coord_max)
        y = rd.randrange(0,coord_max)
        if not ([x,y] in pts):
            pts += [[x,y]]
    return pts

def generer_ch_aléatoire(n):
    presents = []
    manquants = [True]*n
    while len(presents) < n:
        tmp = rd.randint(0,n-1)
        if manquants[tmp]:
            manquants[tmp] = False
            presents.append(tmp)
    return presents
```

ANNEXE

```python
def tabl_dist(pts):                                    # Matrice symetrique contenant les distance entre les différents points
    n = len(pts)
    tableau = np.zeros((n,n))
    for i in range(n):
        for j in range(i+1,n):
            dist = distance(pts[i],pts[j])
            tableau[i,j] = dist
            tableau[j,i] = dist
    return tableau

def longueur_ch(ch,tableau):                           # Calculer la longueur d'un chemin
    long = 0
    p0 = len(tableau)-1
    for p1 in ch:
        long += tableau[p0,p1]
        p0 = p1
    return long
```

```python
## ALGORITHME GLOUTON

def algo_glouton(pts):                                 # Generer un chemin par la éthode du plus proche voisin
    n =len(pts)
    tableau = tabl_dist(pts)
    long_opt = np.inf
    ch_opt = []
    for k in range(n):
        visités = [k]
        manquants = [True]*n
        manquants[k] = False
        for l in range(n-1):
            dist_proche = np.inf
            for t in range(n):
                d = tableau[visités[l],t]
                if d <= dist_proche and manquants[t]:
                    dist_proche = d
                    indice = t
            visités.append(indice)
            manquants[indice] = False
        long = longueur_ch(visités,tableau)
        if long <= long_opt:
            long_opt = long
            ch_opt = visités
    return [long_opt,ch_opt]

```

ANNEXE

```
73 ##
74
75 def fusion(l1,l2):
76     if len(l1)==0:
77         return l2
78     if len(l2)==0:
79         return l1
80     if l1[0][0]>=l2[0][0]:
81         return [l2[0]]+fusion(l1,l2[1::])
82     else:
83         return [l1[0]]+fusion(l1[1::],l2)
84
85 def separe(l):
86     if len(l)<2:
87         return l
88     else:
89         u=l[:(len(l)//2)]
90         v=l[len(l)//2:]
91         return [u,v]
92
93 def tri_fusion(l):
94     if len(l)<2:
95         return l
96     else:
97         [u,v]=separe(l)
98         return fusion(tri_fusion(u),tri_fusion(v))
```

ANNEXE

```python
## ALGORITHME GENETIQUE

def creer_population(m,tableau):
    population = []
    ch = list(range(len(tableau)))
    for i in range(m):
        rd.shuffle(ch)
        longueur = longueur_ch(ch,tableau)
        population.append([longueur,ch])
    return population

def reduire(population):
    l = tri_fusion(population)
    population[:] = l[:len(population)//2]

def normaliser_ch(ch,n):
    presents = []
    manquants = [True]*n
    for p in ch:
        if p < n and manquants [p]:
            presents.append(p)
            manquants[p] = False
    for i in range(n):
        if manquants[i]:
            presents.append(i)
    return presents
```

ANNEXE

```
126
127 def muter_ch(ch):
128     n = len(ch)
129     i = j = rd.randrange(0,n)
130     while j == i:
131         j = rd.randrange(0,n)
132     ch[i],ch[j] = ch[j],ch[i]
133
134 def muter_population(population,proba,tableau):
135     for i in range(1,len(population)):
136         if rd.random() < proba:
137             ch = population[i][1]
138             muter_ch(ch)
139             population [i] = [longueur_ch(ch,tableau),ch]
140
141 def croiser(c1,c2):
142     n = len(c1)
143     return normaliser_ch(c1[:n//2]+c2[n//2:],n)
144
145 def nouvelle_generation(population,tableau):
146     n = len(population)
147     for i in range(n-1):
148         ch = croiser(population[i][1],population[i+1][1])
149         population.append([longueur_ch(ch,tableau),ch])
150     ch = croiser(population[0][1],population[n-1][1])
151     population.append([longueur_ch(ch,tableau),ch])
152
153 def algo_genetique(pts,nb_pop,proba,gen):
154     tableau = tabl_dist(pts)
155     population = creer_population(nb_pop,tableau)
156     for i in range(gen):
157         reduire(population)                                    # Selection
158         nouvelle_generation(population,tableau)                # Croisement
159         muter_population(population,proba,tableau)             # Mutation
160     return tri_fusion(population)[0]
```

ANNEXE

```
## EXPERIENCES

def trace_compare(pts,ch1,ch2):
    tabl = tabl_dist(pts)
    n = len(ch1)
    x1=[]
    y1=[]
    x2=[]
    y2=[]
    for j in range(n):
        k1=ch1[j]
        x1+=[pts[k1][0]]
        y1+=[pts[k1][1]]
        k2=ch2[j]
        x2+=[pts[k2][0]]
        y2+=[pts[k2][1]]

    plt.figure()
    plt.subplot(1,2,1)
    plt.scatter(x1,y1,color='red')
    plt.plot(x1,y1,label=longueur_ch(ch1,tabl))
    plt.title('parcours sur chemin aléatoire 1')   # à completer le titre
    plt.legend()
    plt.grid()


    plt.subplot(1,2,2)
    plt.scatter(x2,y2,color='red')
    plt.plot(x2,y2,label=longueur_ch(ch2,tabl))
    plt.title('parcours sur chemin aléatoire 2')   # à completer le titre
    plt.legend()
    plt.grid()

    plt.show()
```

ANNEXE

```
197 def trace_chemin(pts,ch):
198     tabl = tabl_dist(pts)
199     n = len(ch)
200     x=[]
201     y=[]
202     for j in range(n):
203         k=ch[j]
204         x+=[pts[k][0]]
205         y+=[pts[k][1]]
206     plt.scatter(x,y,color='red')
207     plt.plot(x,y,label=longueur_ch(ch,tabl))
208     plt.title("chemin construit par l'algorithme glouton")                    # à completer le titre
209     plt.legend()
210     plt.grid()
211     plt.show()
212 
213 def experience(liste,m):
214 
215     Y_plus = []
216     Y_moy = []
217     Y_moins = []
218     moy = 0
219     for k in range(len(liste)):
220         print(k)
221         moy = 0
222         min = np.inf
223         max = -np.inf
224         for i in range(m):
225             tmp = algo_glouton(generer_pts(liste[k],m))[0]
226             moy += tmp
227             if max<tmp:
228                 max = tmp
229             if min>tmp:
230                 min =tmp
231         Y_plus += [max]
232         Y_moy += [moy/m]
233         Y_moins += [min]
234 
235     plt.plot(liste,Y_plus,color='chartreuse')
236     plt.plot(liste,Y_moy,label = 'dist moy glt',color = 'red')
237     plt.plot(liste,Y_moins,color='chartreuse')
238     plt.fill_between(liste, Y_plus, Y_moins, color='turquoise')
239     plt.grid()
240     plt.legend()
241     plt.show()
```

ANNEXE

```
def regres(l1,l2,n):
    coef = np.polyfit(l1,l2,n)
    f = np.poly1d(coef)
    return coef,f

def experience_reg(liste,m):

    Y_plus = []
    Y_moy = []
    Y_moins = []
    moy = 0
    for k in range(len(liste)):
        print(liste[k])
        moy = 0
        min = np.inf
        max = -np.inf
        for i in range(m):
            tmp = algo_glouton(generer_pts(liste[k],m))[0]
            moy += tmp
            if max<tmp:
                max = tmp
            if min>tmp:
                min =tmp
        Y_plus += [max]
        Y_moy += [moy/m]
        Y_moins += [min]

    pts = liste
    moy_pts = Y_moy
    reg = regres(pts,moy_pts,1)
    func = reg[1]

    x = np.linspace(pts[0],pts[len(pts)-1],5*len(pts))
    y = func(x)

    plt.plot(liste,Y_plus,color='chartreuse')
    plt.plot(liste,Y_moins,color='chartreuse')
    plt.fill_between(liste, Y_plus, Y_moins, color='turquoise')

    plt.plot(x,y,color='black',label='regression')
    plt.plot(pts,moy_pts,'x',color='red',label='points moyens')

    plt.grid()
    plt.legend()
    plt.show()
```

ANNEXE

```
291 def algo_genetique_modifie(pts,nb_pop,proba,gen):
292     l = []
293     tableau = tabl_dist(pts)
294     population = creer_population(nb_pop,tableau)
295     for i in range(gen):
296         reduire(population)
297         l+=[population[0][0]]
298         nouvelle_generation(population,tableau)
299         muter_population(population,proba,tableau)
300     reduire(population)
301     l+=[population[0][0]]
302     return l
303 
304 def trace_algo_gen_convergence(ens_pts,nb_pop,proba,gen):
305     absc = list(range(1,gen+2))
306     genetique = []
307     for ens in ens_pts:
308         ord = algo_genetique_modifie(ens,nb_pop,proba,gen)
309         genetique.append(ord)
310         print(len(absc)) #
311         print(len(ord)) #
312         plt.plot(absc,ord,label =(len(ens), 'points'))
313     plt.legend()
314     plt.title('Concergence des generations')
315     plt.grid()
316     plt.show()
```

ANNEXE

```
368
369 def glouton_vs_genetique(liste,m,pts,nb_population,proba,gen):
370
371     Y_plus = []
372     Y_moy = []
373     Y_moins = []
374
375     Y_genetique = []
376
377     moy = 0
378     for k in range(len(liste)):
379         print(liste[k])
380         moy = 0
381         min = np.inf
382         max = -np.inf
383         for i in range(m):
384             pts = generer_pts(liste[k],100)
385             tmp = algo_glouton(pts)[0]
386             moy += tmp
387             if max<tmp:
388                 max = tmp
389             if min>tmp:
390                 min =tmp
391         Y_plus += [max]
392         Y_moy += [moy/m]
393         Y_moins += [min]
394
395         Y_genetique += [algo_genetique(pts,nb_population,proba,gen)[0]]
396
397     plt.plot(liste,Y_plus,color='chartreuse')
398     plt.plot(liste,Y_moins,color='chartreuse')
399     plt.fill_between(liste, Y_plus, Y_moins, color='turquoise')
400
401     plt.plot(liste,Y_moy,label = 'distance_glouton',color = 'red')
402     plt.plot(liste,Y_genetique,label = 'distance_genetique',color = 'black')
403
404
405
406     plt.grid()
407     plt.legend()
408     plt.show()
409
```

ANNEXE

```
328
329
330 trace_chemin(pts,chemin_aléatoire1)
331
332
333 chemin_aléatoire1 = [32, 28, 96, 12, 78, 60, 62, 98, 46, 67, 14, 55, 83, 50, 74, 71, 4, 53, 65, 80, 70, 35, 33, 79, 30, 40, 38, 76, 7, 84, 19,
    43, 11, 5, 24, 66, 44, 1, 20, 17, 51, 82, 58, 47, 92, 25, 10, 16, 18, 68, 48, 97, 15, 93, 49, 45, 0, 52, 77, 89, 95, 31, 13, 2, 59, 42, 6, 29,
    36, 57, 90, 64, 34, 94, 56, 69, 3, 41, 99, 21, 73, 85, 75, 37, 39, 61, 23, 91, 88, 22, 87, 54, 81, 72, 27, 86, 8, 63, 26, 9]
334
335 chemin_aléatoire2 = [92, 64, 5, 65, 54, 21, 31, 75, 63, 73, 82, 99, 51, 30, 1, 36, 94, 47, 7, 48, 45, 84, 19, 49, 98, 70, 85, 81, 50, 3, 71,
    32, 25, 97, 57, 24, 93, 44, 58, 13, 80, 91, 69, 27, 8, 34, 88, 53, 90, 38, 20, 29, 23, 0, 33, 87, 37, 56, 68, 17, 78, 11, 83, 39, 89, 9, 2, 43,
    18, 40, 4, 6, 55, 60, 61, 15, 12, 86, 59, 67, 76, 46, 52, 10, 72, 74, 41, 16, 95, 14, 26, 62, 77, 42, 96, 66, 35, 22, 79, 28]
336
337 pts = [[93, 14], [26, 89], [52, 99], [48, 90], [6, 11], [39, 89], [0, 40], [86, 85], [29, 68], [79, 3], [97, 18], [53, 19], [94, 32], [0, 1],
    [89, 5], [69, 41], [19, 73], [49, 38], [10, 80], [87, 31], [35, 3], [19, 28], [60, 14], [60, 42], [0, 13], [95, 5], [7, 9], [74, 0], [35, 30],
    [10, 40], [84, 72], [9, 79], [45, 78], [70, 95], [17, 80], [26, 24], [92, 65], [16, 69], [1, 16], [69, 73], [38, 77], [1, 51], [80, 35], [72,
    92], [4, 86], [87, 65], [75, 76], [46, 13], [61, 7], [22, 8], [72, 5], [61, 2], [66, 83], [90, 13], [46, 75], [85, 7], [34, 88], [74, 29], [61,
    16], [33, 98], [99, 6], [43, 36], [56, 99], [72, 97], [70, 16], [90, 48], [55, 0], [37, 45], [20, 34], [55, 52], [14, 66], [22, 45], [63, 51],
    [47, 45], [28, 41], [71, 60], [80, 40], [1, 22], [7, 98], [73, 1], [6, 46], [60, 24], [88, 4], [0, 8], [25, 96], [64, 28], [31, 74], [31, 71],
    [57, 22], [66, 33], [74, 70], [19, 63], [46, 77], [2, 80], [69, 50], [78, 16], [1, 20], [99, 13], [29, 96], [39, 2]]
338
339 trace_chemin(pts,chemin_aléatoire1)
340
341 proba = 0.5
342 nb_population = 300
343 gen = 10000
344
345 liste = list(range(20,100))
346 m = 100
347 experience_reg(liste,m)
348
```