Problème de Satisfaction de contrainte :

Dans ce use case, nous avons modélisé et résolu le problème de satisfaction de contrainte avec 3 algorithmes de search : backtracking_search , forward_checking_search et mac_search.

Le problème défini par le set de paramètre (n = 10 , alpha = 0.9 , r = 0.4) est défini comme suit :

10 variables, dont chacune est défini sur un domaine de taille d = n^alpha = 8

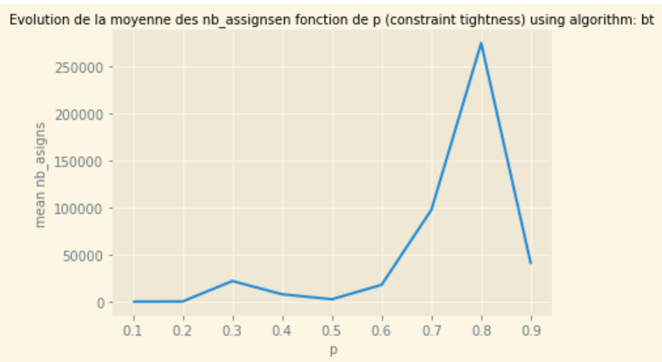
Pour 9 valeurs de constraint tightness (p) [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] , on simule le problème 20 fois vu l'aspect aléatoire des algorithmes ; pour une seule valeur de p , on résout le problème 20 fois

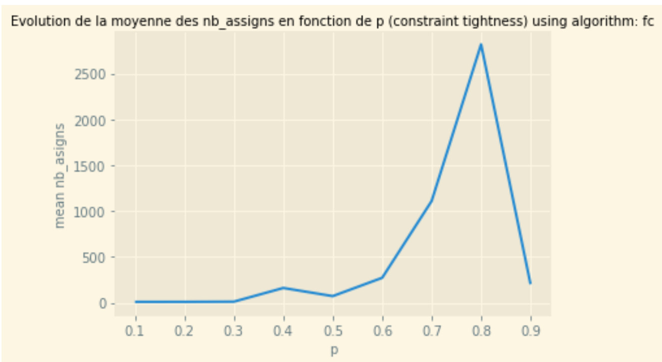
- Exploitation et Analyse des résultats obtenus de la simulation :

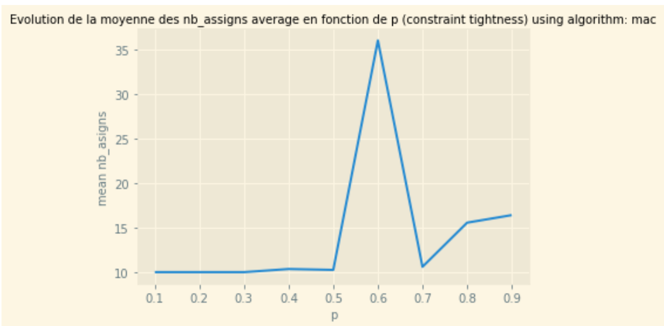
L'output du modèle est un data frame qui résume le nb_assigns permettant d'assigner tous les variables en utilisant chaque modèle de search pour chaque valeur de p 20 fois.

1. Etude de la moyenne de nb_assigns :

Evolution de la moyenne de nb_assigns en fonction de p par les trois algorithmes :







On remarque que quel que soit le modèle de search, la moyenne de nb_assigns est a presque le même comportement : régime stable – croissance rapide - chute

Pour une constraint thightness dans l'intervalle [0.1 , 0.5], la moyenne de nb_assigns est presque constante (10 assigns en moyenne) pour le modèle mac par contre pour les deux autres bt et fc on note une augmentation brusque de nb_assigns en moyenne entre [0.2, 0.3] et [0.3, 0.4] respectivement.

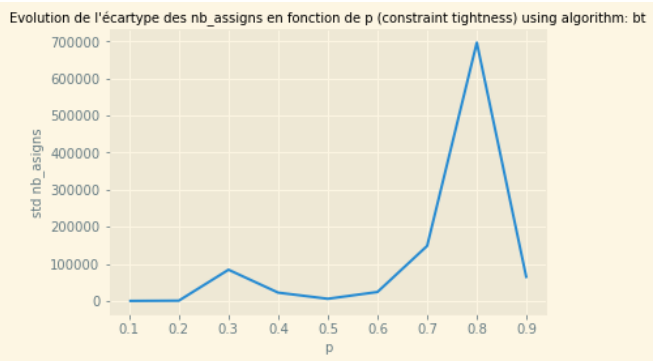
Par le modèle mac, la moyenne de nb_assigns commence à augmenter dès le 5ème p ; p=0.5 pour atteindre une valeur de 37 nb_assigns en moyenne pour ensuite baisser jusqu'à atteindre la valeur de 10 et augmenter à nouveau

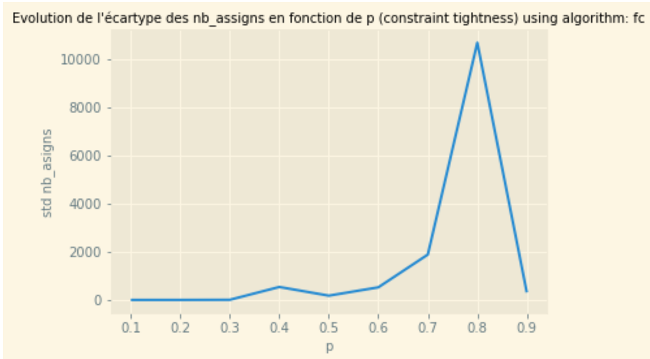
Par les modèles bt et fc, la moyenne de nb_assigns commencer à augmenter dès le 6ème p ; p=0.6 pour atteindre une valeur maximale de 250000 et 2500 respectivement ;

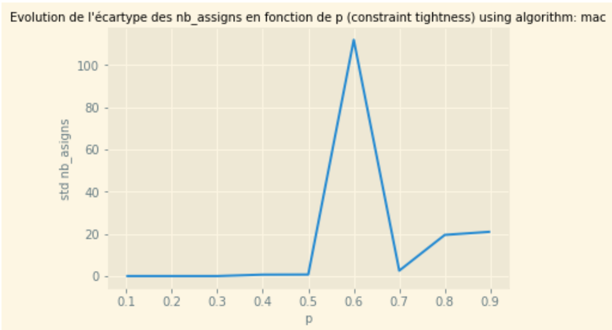
$pic_{bt} = 100 * pic_{bt}$, pour ensuite baisser.

2. Etude de l'écart type de nb_assigns :

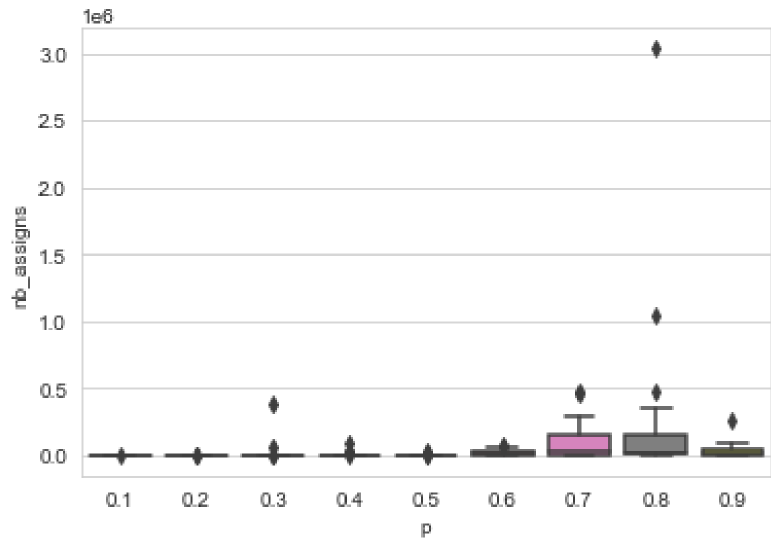
L'écartype du résultat nb_assigns à le même comportement que la moyenne par les 3 modèles. On note de grands écarts types pour les valeurs de p 0.7, 0.8 et 0.9 pour bt et fc ce qui signifie qu' en résolvant le même csp 20 fois avec les mêmes paramètres on obtient des valeurs très dispersés.







ON remarque que les petites valeurs de p ne sont pas sensibles à l'aspect aléatoire de la simulation c'est-à-dire que pour une p petite on a le même résultat nb_assigns en répétant l'experience 20 fois par contre pour des valeurs plus proches de 1, on note une non stabilité des résultats et même des points abhérents.



Annexe :

- Après avoir résolu le CSP par les trois modèles de search pour 9 valeurs de p 20 fois, on récupère la data results :

results

	instance	algorithm	p	nb_assigns
0	0.1-0	bt	0.10000	11
1	0.1-0	fc	0.10000	10
2	0.1-0	mac	0.10000	10
3	0.1-1	bt	0.10000	12
4	0.1-1	fc	0.10000	10
5	0.1-1	mac	0.10000	10
6	0.1-2	bt	0.10000	11
7	0.1-2	fc	0.10000	10
8	0.1-2	mac	0.10000	10
9	0.1-3	bt	0.10000	13
10	0.1-3	fc	0.10000	10
11	0.1-3	mac	0.10000	10
12	0.1-4	bt	0.10000	10
13	0.1-4	fc	0.10000	10
14	0.1-4	mac	0.10000	10
15	0.1-5	bt	0.10000	12
16	0.1-5	fc	0.10000	10
17	0.1-5	mac	0.10000	10
18	0.1-6	bt	0.10000	11

- On applique un groupby des données sur les colonnes ‘p’ et ‘algorithm’ pour calculer la moyenne et l’écartype de nb_assigns sur l’ensemble de simulations (20) :

results = results[['instance','algorithm','p','nb_assigns']]

data = results.copy()

grouped_data = data.groupby(['p','algorithm']).agg(mean=('nb_assigns', 'mean'),std=('nb_assigns','std')).reset_index()

Data Total:

	p	algorithm	mean	std
0	0.1	bt	11.45	1.145931
1	0.1	fc	10.00	0.000000
2	0.1	mac	10.00	0.000000
3	0.2	bt	214.15	677.784182
4	0.2	fc	10.25	0.638666
5	0.2	mac	10.00	0.000000
6	0.3	bt	21899.15	84075.225989
7	0.3	fc	12.20	5.287523
8	0.3	mac	10.00	0.000000
9	0.4	bt	7800.85	22295.644329

Plot du graphe avec la librairie matplotlib: Les graphes sont de chaque algorithme sont tracés séparément vu la grande différence d’échelle.

```
import matplotlib.pyplot as plt
import numpy as np

algorithms = list(grouped_data['algorithm'].unique())
with plt.style.context('Solarize_Light2'):
    plt.plot(grouped_data[grouped_data['algorithm']==algorithms[0]]['p'],
             grouped_data[grouped_data['algorithm']==algorithms[0]]['mean'] )
    plt.title('Evolution de la moyenne des nb_assignsen fonction de p (constraint tightness)
               using algorithm: '+ algorithms[0], fontsize=10)
    plt.xlabel('p', fontsize=10)
    plt.ylabel('mean nb_asigns', fontsize=10)
plt.show()
```

Devoir code:

```
from random_csp import *
import pandas as pd
import numpy as np
n = 10
alpha = 0.9
r = 0.4
# we fix p to 0.6 as example
p = 0.5
# we generate a random csp

csp = generate_csp(n, p, alpha, r)
print(csp)
# in the output, you will see a constraint like
# TableConstraint([0, 1]):
# incompatibles values
# [(4, 4), (4, 5), (6, 6), (1, 1), (6, 2), (4, 2), (4, 1), (5, 5), (7,
0),
# (5, 7), (3, 0), (7, 6), (5, 4), (0, 4), (5, 6), (6, 0), (3, 7), (7,
7),
# (2, 2), (4, 3), (4, 6), (0, 5), (0, 2), (2, 3), (5, 6), (4, 0), (3,
0),
# (3, 1), (1, 2), (7, 4), (7, 0), (3, 4), (7, 3)]
# which means that for variable 0 and 1
# (4, 4), (4, 5) ... are not allowed by this constraint, we vca assign
for example
# variable 0 to  1 and variable 1  to  3, 4, 5, 6, or 7
# e.g  this constraint is  is satisfied by the assignment {0:1, 1:3}

# We launch solution search on copies of the problem
# to compare the three algorithms on the same problems
p1 = copy.deepcopy(csp)
print(backtracking_search(p1))
print(p1.nb_assigns)

p2 = copy.deepcopy(csp)
print(forward_checking_search(p2))
print(p2.nb_assigns)

p3 = copy.deepcopy(csp)
print(mac_search(p3))
print(p3.nb_assigns)

# Work to do
# for each value of p from [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
0.9]
# you must generate (as we do in the test before) 20 instances of csp
# and test the three algorithms on copies of these instances.
# The results (number of assignment made by each algorithm) must be
collected,
# plotted and discussed in your report

# you can use the following program  :

columns = ['instance','algorithm','p', 'nb_assigns']

results = pd.DataFrame(columns=columns)# initialized to zeros
n ,alpha, r = 10,  0.9,  0.4
for p in [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]:
    for i in range(20):
```

```
        csp = generate_csp(n, p, alpha, r)
        csp1 = copy.deepcopy(csp)
        backtracking_search(csp1)
        row_to_append = [str(p)+'-'+ str(i), 'bt', p,
csp1.nb_assigns]
        results.loc[len(results)] = row_to_append

        csp2 = copy.deepcopy(csp)
        forward_checking_search(csp2)
        row_to_append = [str(p)+'-'+ str(i), 'fc', p,
csp2.nb_assigns]
        results.loc[len(results)] = row_to_append

        csp3 = copy.deepcopy(csp)
        mac_search(csp3)
        row_to_append = [str(p)+'-'+ str(i), 'mac', p,
csp3.nb_assigns]
        results.loc[len(results)] = row_to_append


print('-')
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
data = results.copy()
grouped_data = data.groupby(['p','algorithm']).agg(mean=('nb_assigns',
'mean'),std=('nb_assigns','std')).reset_index()

grouped_data[:]
grouped_data.columns


algorithms = list(grouped_data['algorithm'].unique())
with plt.style.context('Solarize_Light2'):

plt.plot(grouped_data[grouped_data['algorithm']==algorithms[0]]['p'],

grouped_data[grouped_data['algorithm']==algorithms[0]]['mean'] )
    plt.title('Evolution de la moyenne des nb_assignsen fonction de p
(constraint tightness)
              using algorithm: '+ algorithms[0], fontsize=10
             )
    plt.xlabel('p', fontsize=10)
    plt.ylabel('mean nb_asigns', fontsize=10)
plt.show()



algorithms = list(grouped_data['algorithm'].unique())
with plt.style.context('Solarize_Light2'):

plt.plot(grouped_data[grouped_data['algorithm']==algorithms[1]]['p'],g
rouped_data[grouped_data['algorithm']==algorithms[1]]['mean'] )
    plt.title('Evolution de la moyenne des nb_assigns en fonction de p
(constraint tightness) using algorithm: '+ algorithms[1], fontsize=10)
    plt.xlabel('p', fontsize=10)
    plt.ylabel('mean nb_asigns', fontsize=10)
plt.show()


algorithms = list(grouped_data['algorithm'].unique())
```

```python
with plt.style.context('Solarize_Light2'):

plt.plot(grouped_data[grouped_data['algorithm']==algorithms[2]]['p'],g
rouped_data[grouped_data['algorithm']==algorithms[2]]['mean'] )
    plt.title('Evolution de la moyenne des nb_assigns en fonction de p
(constraint tightness) using algorithm: '+ algorithms[2], fontsize=10)
    plt.xlabel('p', fontsize=10)
    plt.ylabel('mean nb_asigns', fontsize=10)
plt.show()

grouped_data[grouped_data['algorithm']==algorithms[0]]


algorithms = list(grouped_data['algorithm'].unique())
with plt.style.context('Solarize_Light2'):

plt.plot(grouped_data[grouped_data['algorithm']==algorithms[0]]['p'],g
rouped_data[grouped_data['algorithm']==algorithms[0]]['std'] )
    plt.title("Evolution de l'écartype des nb_assigns en fonction de p
(constraint tightness) using algorithm: "+ algorithms[0], fontsize=10)
    plt.xlabel('p', fontsize=10)
    plt.ylabel('std nb_asigns', fontsize=10)
plt.show()

algorithms = list(grouped_data['algorithm'].unique())
with plt.style.context('Solarize_Light2'):

plt.plot(grouped_data[grouped_data['algorithm']==algorithms[1]]['p'],g
rouped_data[grouped_data['algorithm']==algorithms[1]]['std'] )
    plt.title("Evolution de l'écartype des nb_assigns en fonction de p
(constraint tightness) using algorithm: "+ algorithms[1], fontsize=10)
    plt.xlabel('p', fontsize=10)
    plt.ylabel('std nb_asigns', fontsize=10)
plt.show()

algorithms = list(grouped_data['algorithm'].unique())
with plt.style.context('Solarize_Light2'):

plt.plot(grouped_data[grouped_data['algorithm']==algorithms[2]]['p'],g
rouped_data[grouped_data['algorithm']==algorithms[2]]['std'] )
    plt.title("Evolution de l'écartype des nb_assigns en fonction de p
(constraint tightness) using algorithm: "+ algorithms[2], fontsize=10)
    plt.xlabel('p', fontsize=10)
    plt.ylabel('std nb_asigns', fontsize=10)
plt.show()
import seaborn as sns
sns.boxplot(x = 'algorithm', y = 'nb_assigns', data = results)

sns.set_style("whitegrid")

sns.boxplot(x='p', y='nb_assigns', data=results[results['algorithm']
== algorithms[0]])
```