PROJET DE RÉSOLUTION DE PROBLÈMES :

*METAHEURISTIQUES POUR LA RÉSOLUTION DU PROBLÈME DE L'**ARBRE DE STEINER** DE POIDS MINIMUM*

Introduction

Le problème de l'arbre de Steiner est un problème d'optimisation combinatoire dont nous nous appliquons, dans le cadre de ce projet, à résoudre la version graphique. Elle consiste à rechercher dans un graphe connexe valué le sous-ensemble de nœuds « steiner » (non-terminaux) permettant d'obtenir un arbre couvrant tous ses nœuds terminaux et qui soit de poids minimum. À cette fin, deux métaheuristiques nous sont proposées : l'une dont la progression vers l'optimum global se fait suivant un algorithme génétique et l'autre par simple recherche locale.

Le but du projet est de les comparer. Pour ce faire, dans un premier temps, nous rappelerons les techniques envisagées pour générer des « bonnes solutions » (heuristiques du Plus Court Chemin (PCC) et de l'Arbre Couvant Minimum (ACM)) et analyserons leurs résultats (valeurs trouvées) et performances (temps de calcul) sur les différentes instances fournies. Puis, nous évoquerons les principes des métaheuristiques proposées et analyserons les résultats de leur application sur ces mêmes instances.

A. HEURISTIQUES POUR LE CALCUL DE BONNES SOLUTIONS

Dans cette section, nous comparerons suivant deux (02) critères (valeur de la solution et temps de calcul) les heuristiques du plus court chemin et de l'arbre couvrant minimum. Mais avant, nous en rappelons brièvement les principes.

1. Heuristique du Plus Court Chemin (PCC)
   - Principe

Son fonctionnement s'articule en cinq (05) étapes :

Étape 1 : La construction d'un graphe G1 de distances entre les sommets terminaux de G.

Étape 2 : Le calcul d'un arbre G2 de poids minimum couvrant les sommets terminaux.

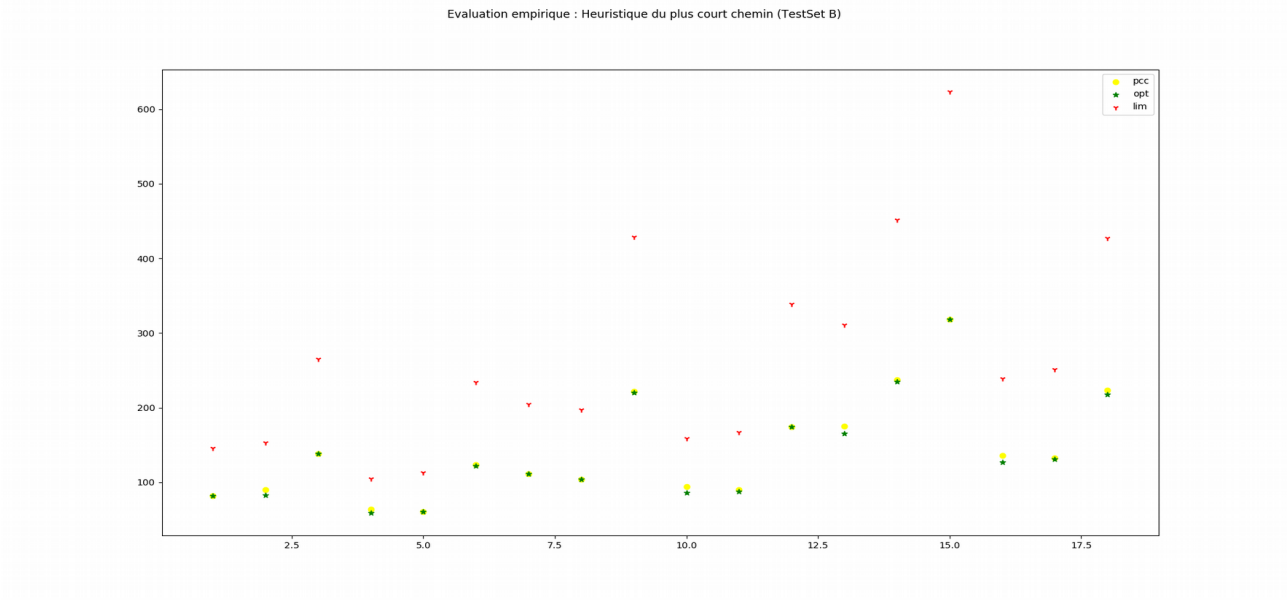
Étape 3 : Construire le sous-graphe G3 de G en remplaçant chaque arête de G2 par les arêtes situés sur le plus court chemin correspondant dans G.

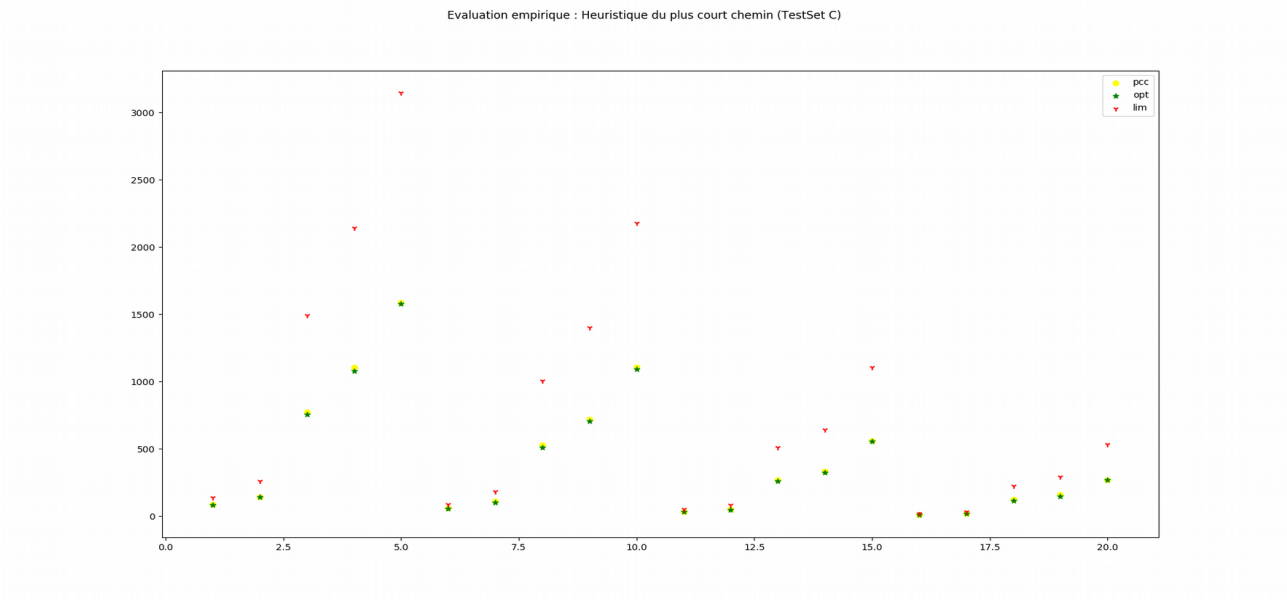
Étape 4 : Constuire l'arbre de poid minimum G4 couvrant les sommets de G3 (supprimant ainsi les arêtes se trouvant sur plus d'un des plus courts chemins considérés)

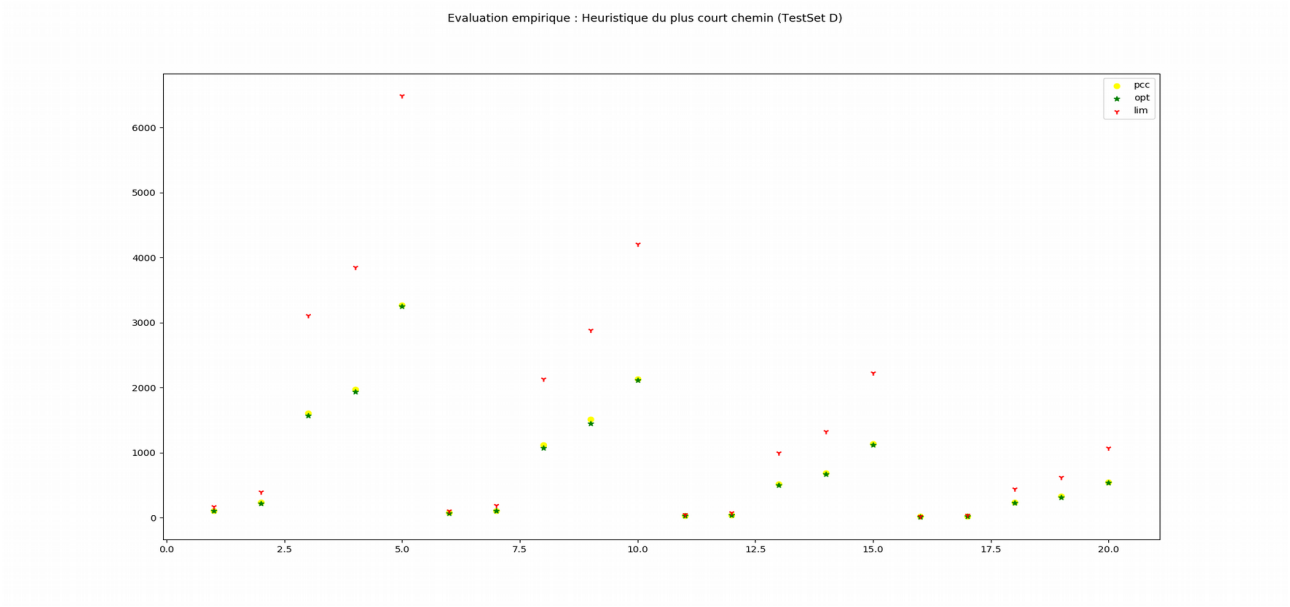
Étape 5 : Éliminer les feuilles non-terminales de G4 pour obtenir G5 : la solution de l'heuristique.

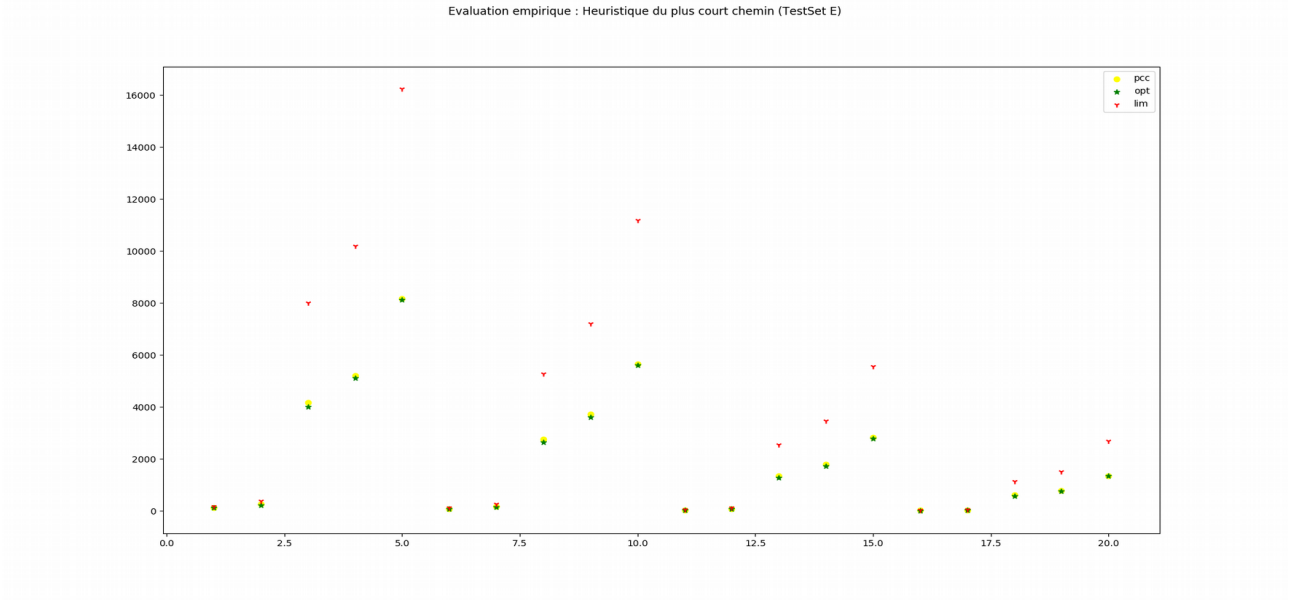
   - Évaluation de la solution obtenue

Les graphiques suivants nous permettent d'apprécier la qualité de la solution retournée par l'algorithme du plus court chemin. Elle semble assez proche de la solution optimale et s'écarte considérablement à la borne supérieure déduite de son rapport d'approximation (2 – 2/ #T) où #T représente le cardinal de l'ensemble des nœuds terminaux de G.


Evaluation empirique : Heuristique du plus court chemin (TestSet B)
pcc
opt
lim
600
500
400
300
200
100
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Evaluation empirique : Heuristique du plus court chemin (TestSet C)
pcc
opt
lim
3000
2500
2000
1500
1000
500
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0





2. Heuristique de l'Arbre Couvrant minimum (ACM)
   - Principe

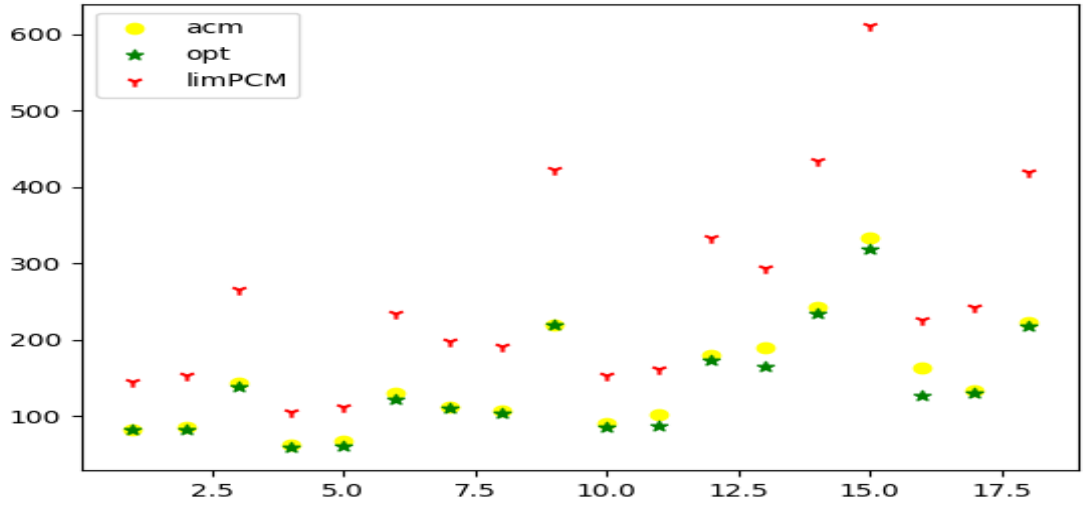
Elle consiste à réduire successivement l'ensemble des nœuds « steiner » par construction d'un arbre couvrant de poids minimum. Les nœuds supprimés à chaque étape sont des nœuds non-terminaux feuilles de l'arbre couvrant calculé. Intuitivement, elle paraît bien moins efficace que l'heuristique du PCC (qui est bien plus élaborée). Le paragraphe suivant en fournit les résultats d'évaluation sur les instances B, C, D et E.

   - Évaluation de la solution obtenue

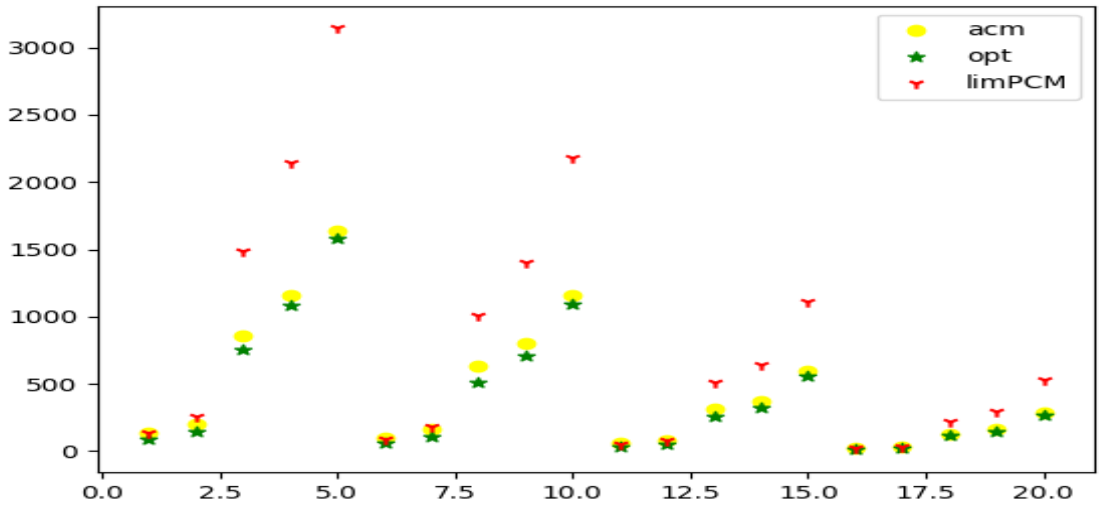
Comme l'indiquent les graphiques ci-dessous, les solutions retournées par cette heuristique sont de loin meilleures à celles que retourneraient des instances critiques de graphe auxquelles l'on appliquerait l'heuristique du PCC. Elles semblent même presque aussi bonnes que celles de cette dernière. Pour y voir plus clair, dans le prochain paragraphe, nous comparons ces deux heuristiques

d’une part, par leur écart par rapport à la solution optimale (en pourcentage) et d’autre part, par leur temps d’exécution.

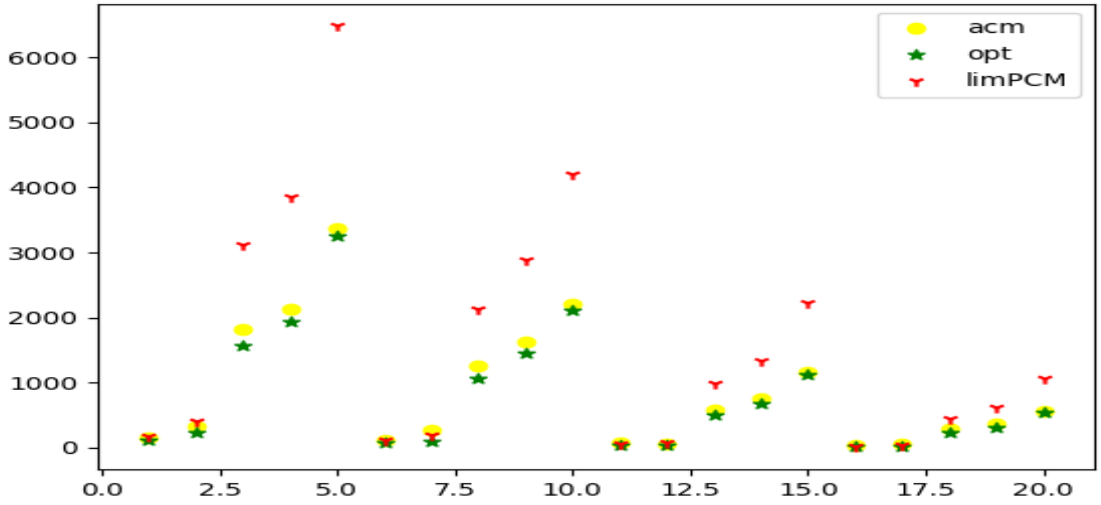
Evaluation empirique : Heuristique de l'arbre couvrant minimum(TestSet B)



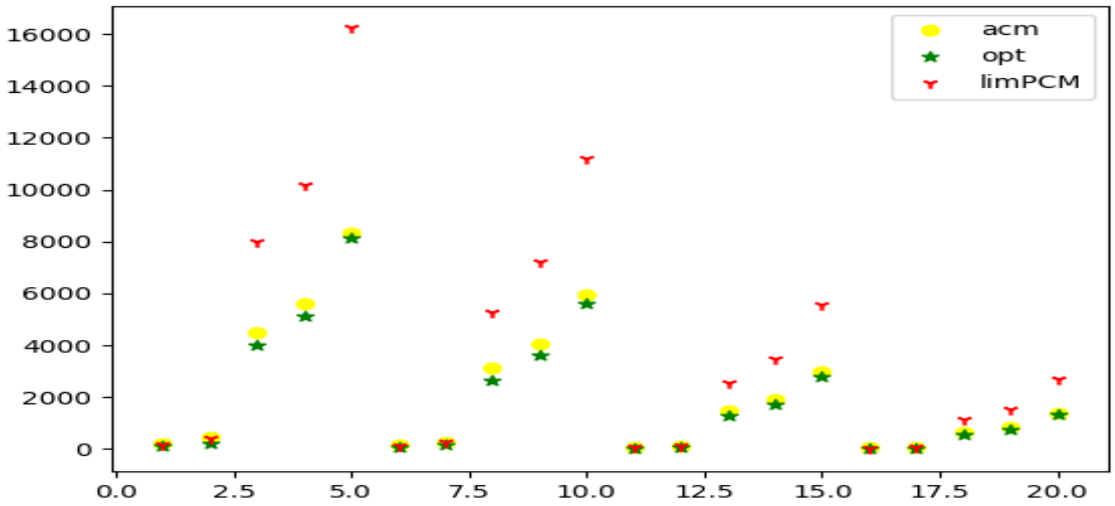
Evaluation empirique : Heuristique de l'arbre couvrant minimum(TestSet C)



Evaluation empirique : Heuristique de l'arbre couvrant minimum(TestSet D)

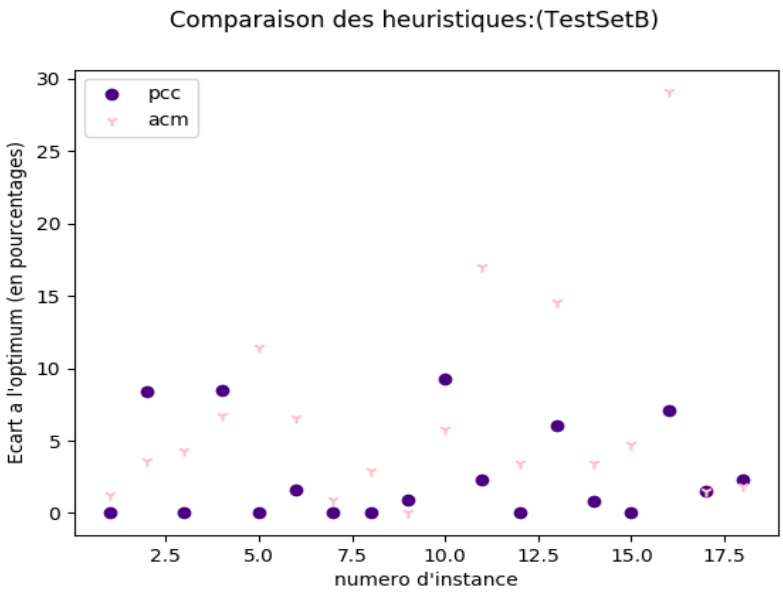


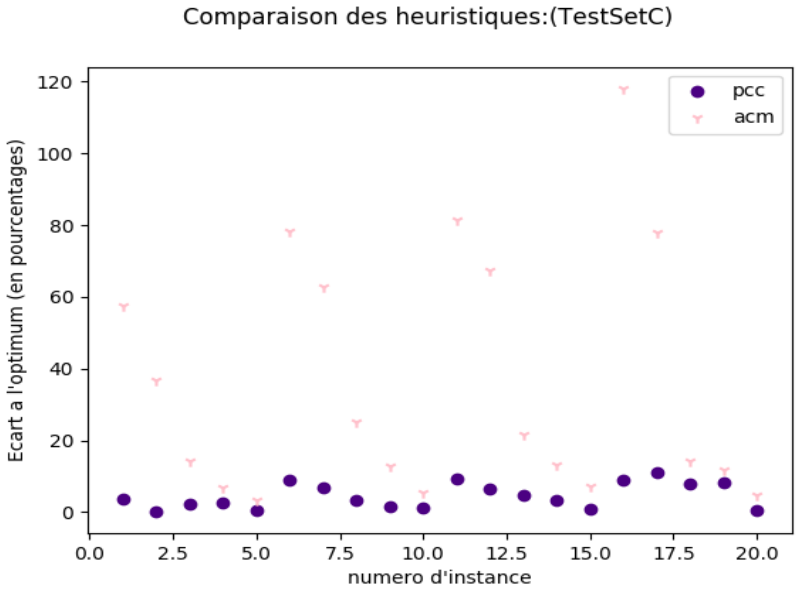
Evaluation empirique : Heuristique de l'arbre couvrant minimum(TestSet E)

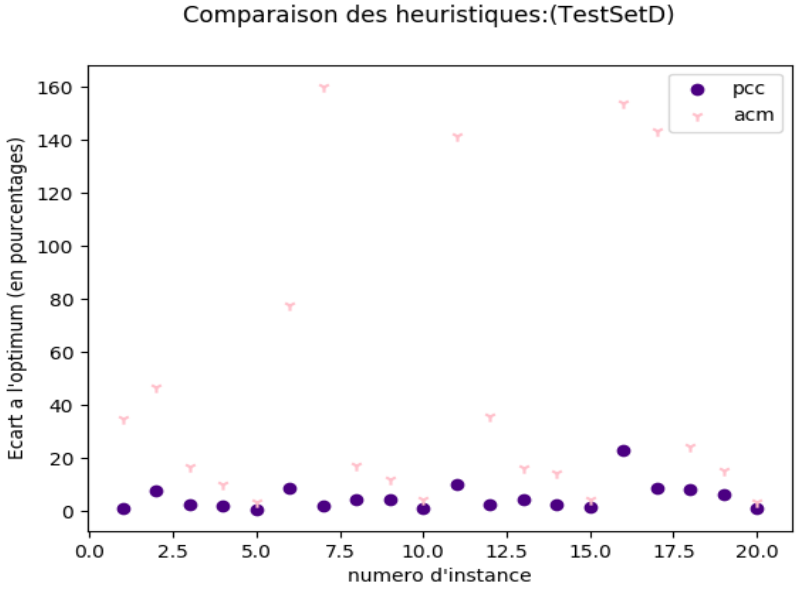


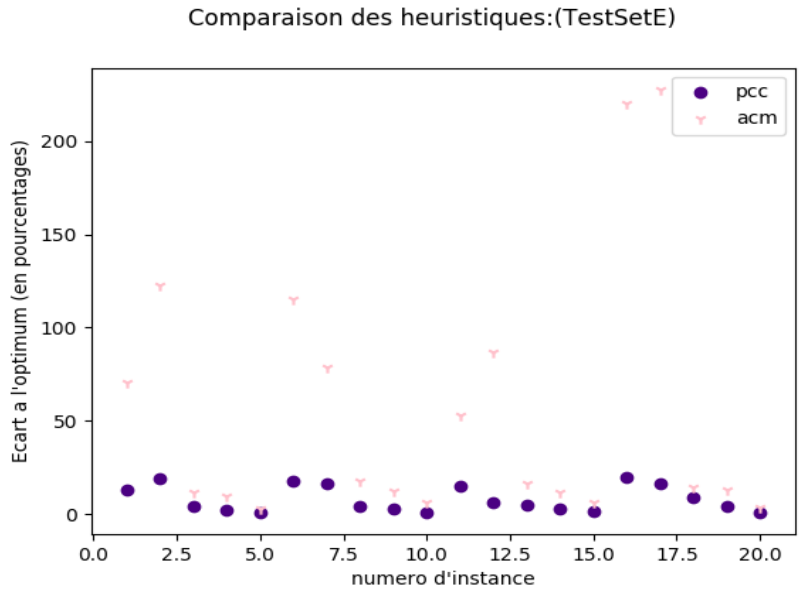
3. PCC Versus ACM
   - Critère : Écart à la solution en pourcentage

Les solutions obtenues par l'heuristique du PCC, quelque soit la classe d'instances considérée s'écarte d'au plus 20 % de la solution optimale tandis que celles de l'heuristique de l'ACM semblent globalement s'accroître avec la taille des instances (voir graphiques ci-dessous). Cette heuristique retourne parfois une solution deux (02) fois supérieure à l'optimum.


Comparaison des heuristiques:(TestSetB)
pcc
acm
Ecart a l'optimum (en pourcentages)
numero d'instance

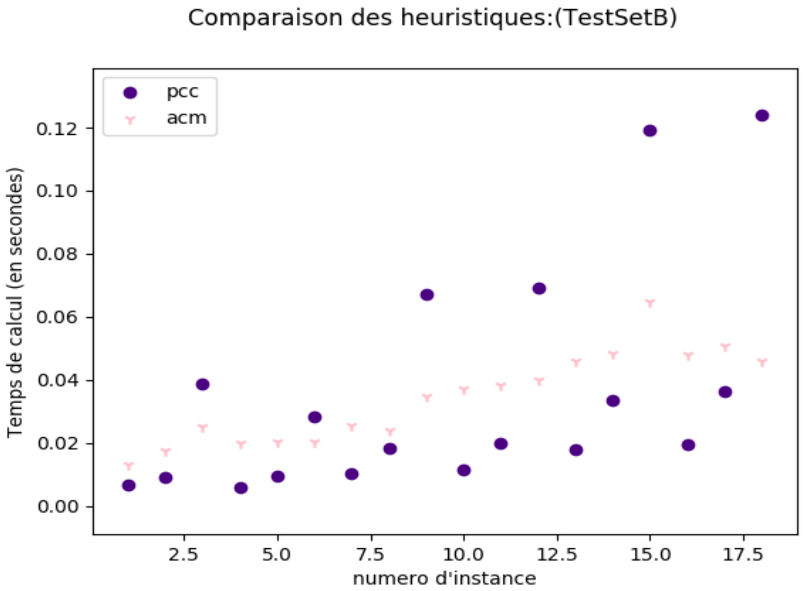

Comparaison des heuristiques:(TestSetC)
pcc
acm
Ecart a l'optimum (en pourcentages)
numero d'instance

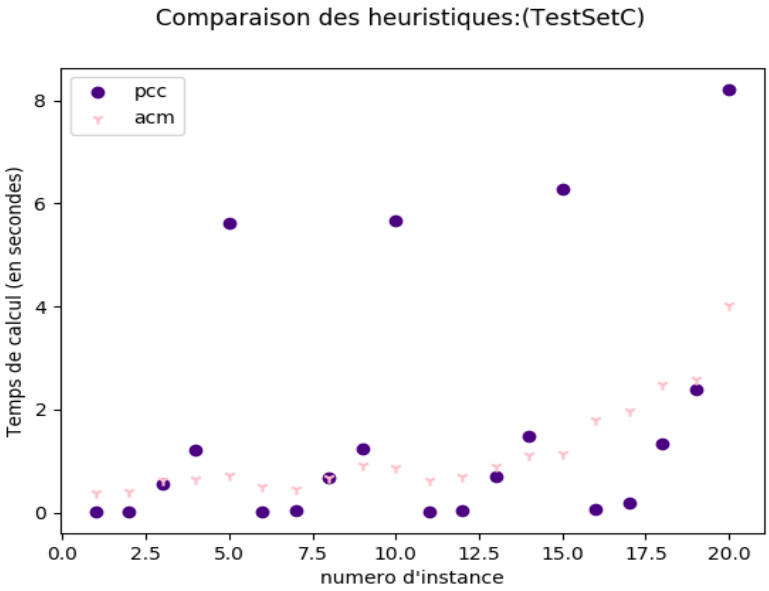

Comparaison des heuristiques:(TestSetD)
pcc
acm
Ecart a l'optimum (en pourcentages)
numero d'instance

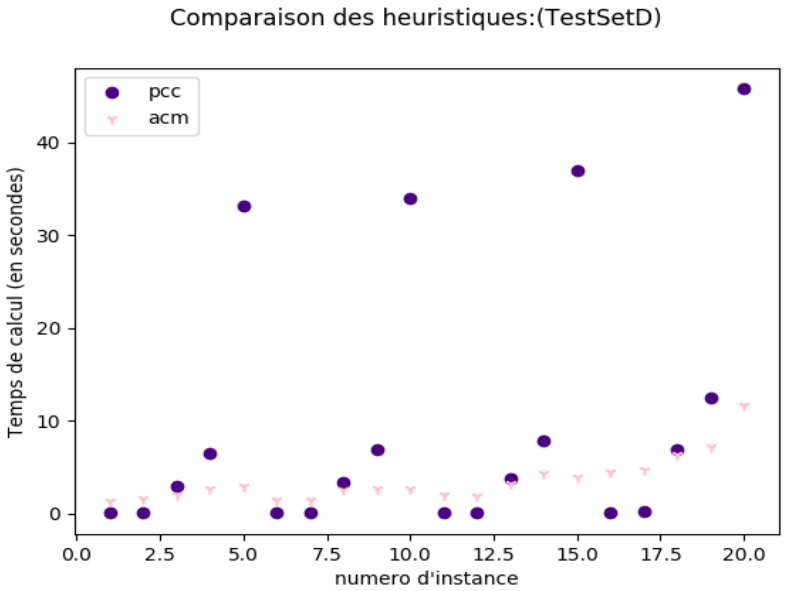


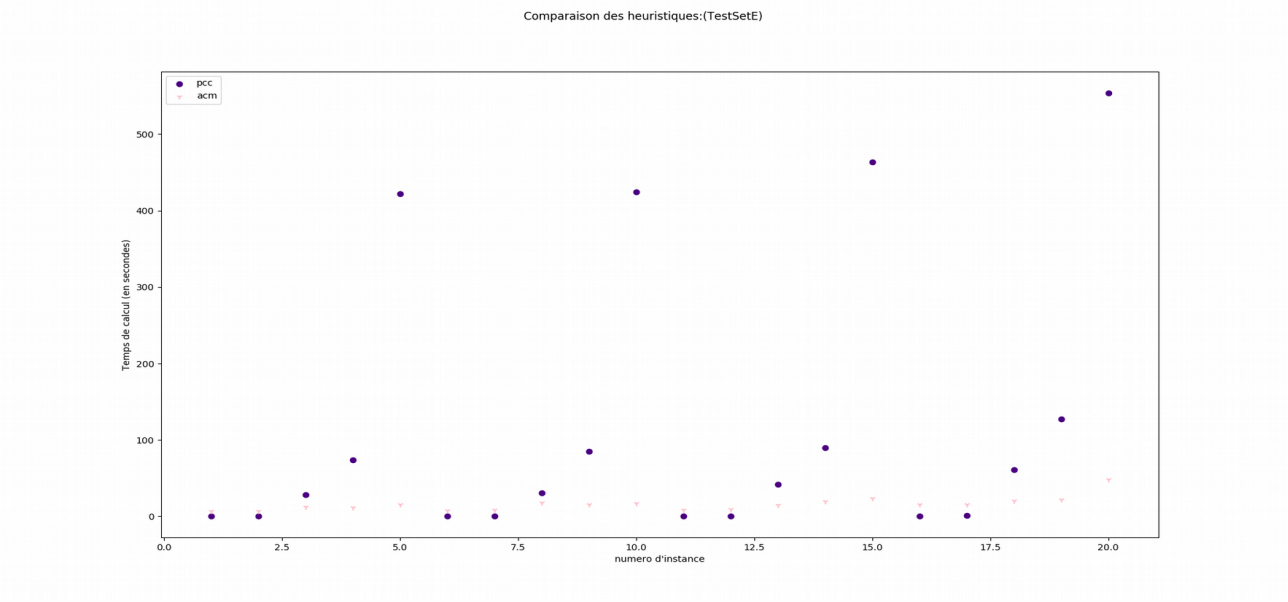
- Critère : temps d'exécution

A contrario, l'efficacité de l'heuristique du PCC est assez coûteuse en temps de calcul. Par exemple, sur la classe d'instances E, le temps de convergence de l'heuristique de l'ACM semble constant (20 secondes) indépendamment de l'instance considérée tandis que le temps d'exécution du PCC croît (de façon importante) avec la taille de l'instance (en particulier avec le nombre de nœuds terminaux). Nous privilégierons donc les solutions retournées par l'heuristique du PCC pour des instances de petites tailles et celle de l'ACM pour des grandes tailles d'instances.




Comparaison des heuristiques:(TestSetC)
pcc
acm
Temps de calcul (en secondes)
numero d'instance
0
2
4
6
8
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Comparaison des heuristiques:(TestSetD)
pcc
acm
Temps de calcul (en secondes)
numero d'instance
0
10
20
30
40
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



B. MÉTAHEURISTIQUES POUR LA RÉSOLUTION DU PROBLÈMES

Comme évoqué dans l'introduction, deux (02) métaheuristiques sont utilisées afin d'améliorer des solutions potentiellement bonnes trouvées à l'aide d'heuristiques. La première est implémentée suivant un algorithme génétique et la seconde une recherche locale. L'objet de cette section est de les comparer du point de vue de l'amélioration qu'elle apporte à la solution retournée par l'heuristique. Rappelons que la population de départ (dans l'algorithme génétique) et les individus-solutions que l'on améliore (dans la recherche locale) sont obtenus par randomisation des heuristiques en bruitant les valeurs des arêtes.
Dans la suite, nous employons le terme individu pour désigner l'ensemble des nœuds « steiner » retenus et fitness pour désigner le poids de l'arbre de Steiner qui résulte de ce choix.

1. Algorithme génétique

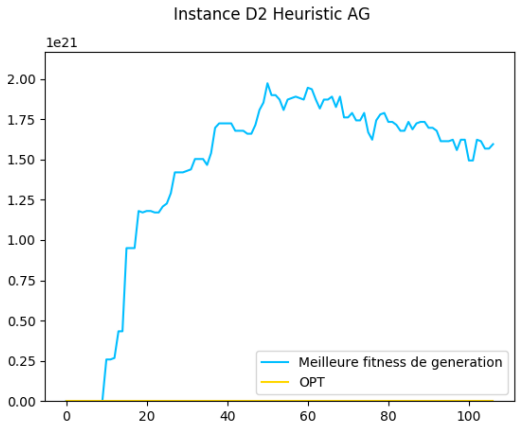
L'implémentation considérée ici se présente comme suit :

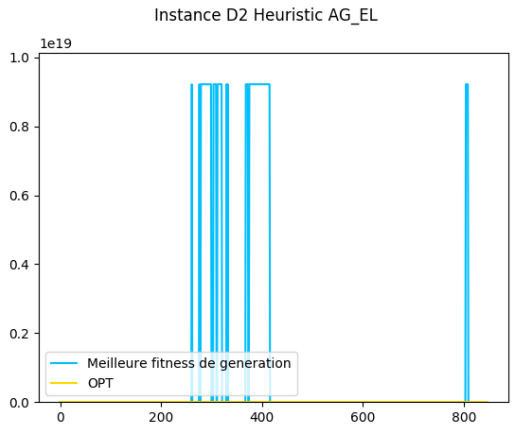
```
gé  né  rer une population initiale P de n individus
faire pour i = 1 à  n
        é  valuer_i pour chaque individu x_i
fait
faire pour i = 1 à  n/2
        sé  lection choisir deux parents x et y dans P en utilisant f
        croisement : avec une proba c é  changer des bits contigus dans x et y
        mutation : avec une probabilité  m, faire muter les bits de x et y
        insertion : insé  rer les nouveaux é  lé  ments obtenus dans P'
fait
P ← P'
retourner l'individu de P de fitness minimum
```

Il convient d'indiquer à ce niveau que la méthode de sélection que nous avons retenue est la méthode de la roulette où l'évaluation considérée pour chaque individu est le complément de sa fitness au cumul des fitness des individus de la population. Supposant par exemple que la population soit composée d'individus de fitness 2, 3, 4 et 7, nous évaluons ces individus à 8, 7, 6 et 9 et en déduisons les probabilités de sélection comme précisé ci-après :

	Individu 1	Individu 2	Individu 3	Individu 4	Total
Fitness	2	3	4	1	10
Évaluation	8	7	6	9	30
Proba. Sélect.	8/30	7/30	6/30	9/30	1

L'opérateur de croisement retenu est l'opérateur de croisement à un point avec c = 0.2.
La probabilité de mutation m a été fixée à 0.05 (plutôt qu'une valeur aléatoire entre 0.02 et 0.04) pour quelque peu « forcer » le renouvellement au fil de générations.
Comme taille de population, nous avons retenu indifféremment de la classe d'instances et de la taille des graphes N = 10 (pour consacrer plus de temps à l'exécution des métaheuristiques).
Pour finir, nous pouvons distinguer deux (02) schémas de remplacement qui sont le remplacement générationnel et le remplacement élitiste. Nous avons retenu le schéma de remplacement élitiste. En effet, comme l'indiquent les deux (02) ci-dessous (qui résument l'évolution de la valeur de la meilleure fitness suivant les deux (02) schémas de remplacement), on remarque que le schéma de remplacement élitiste (à droite) possède de meilleures propriétés « de rappel » vers les solutions bonnes du problème. La comparaison a été effectuée sur l'instance D2 en accordant à chaque algorithme cinq (05) minutes d'exécution.





2. La recherche locale

Ici, l'on calcule le voisin d'un individu en lui ajoutant ou en lui retirant un sommet. Dans ce processus, on retient à chaque fois le premier voisin améliorant.

3. Algorithme génétique Versus Recherche Locale : Les résultats

Les résultats qui suivent se présentent sous forme de tableaux où seront consignés dans l'ordre les résultats de l'évaluation de chaque instance suivant chaque métaheuristique utilisée (valeur VAL, écart à l'optimum POURC_OPT, et temps d'exécution _T). Nous y ajoutons également la valeur minimum entre les résultats retournés par les deux heuristiques (ACM et PCC) sur le graphe originel, l'idée étant de donner une idée de l'amélioration apportée par les métaheuristiques à ces résultats. Le timeout a été choisi égal à 5 minutes. Les individus (de la population de départ ou desquels part la recherche locale) sont générées par randomisation de l'heuristique du PCC pour toutes les instances de la classe B et toutes les instances de la classe C à l'exception des instances C dont les numéros sont multiples de 5. Tous les autres sont générés par randomisation de l'heuristique de l'ACM.

(a) Classe d'instances B

Instances	AG_VAL	POURC_OPT	AG_T	MIN(PCC,ACM)	RL_VAL	POURC_OPT	RL_T	OPT
1	82	0	0	82	82	0	0	82
2	83	0	0	86	83	0	0	83
3	138	0	0	138	138	0	0	138
4	59	0	0	63	59	0	0	59
5	61	0	0	61	61	0	0	61
6	124	2	5	124	122	0	0	122
7	111	0	0	111	111	0	0	111
8	104	0	0	104	104	0	0	104
9	220	0	0	220	220	0	0	220
10	86	0	0,5	91	86	0	0	86
11	90	2	0,7	90	88	0	0,3	88
12	174	0	0	174	174	0	0	174
13	170	3	5	175	168	2	5	165
14	236	0	5	237	235	0	0	235
15	318	0	0	333	318	0	0	318
16	130	2	5	136	127	0	1,2	127
17	131	0	0	133	131	0	0	131
18	218	0	0	222	218	0	0,6	218

Les résultats ci-dessus résument les résultats obtenus après application des deux (02) métaheuristiques : en rose l'algorithme génétique et en violet la recherche locale. Globalement sur cette classe d'instances, la recherche locale est meilleure car elle aboutit à l'obtention de la solution optimale sur toutes les instances sauf l'instance 13.

(b) Classe d'instances C

	AG_VAL	POURC_OPT	AG_T	MIN(PCC,PCM)	RL_VAL	POURC_OPT	RL_T	OPT
1	118	39	5	88	85	0	0	85
2	174	21	5	144	144	0	0	144
3	817	8	5	772	764	1	5	754
4	1153	7	5	1107	1094	1	5	1079
5	1583	0	5	1585	1631	3	5	1579
6	82	49	5	60	55	0	0,2	55
7	147	44	5	109	102	0	0,9	102
8	607	19	5	526	518	2	5	509
9	798	13	5	717	717	1	5	707
10	1105	1	5	1105	1153	5	5	1093
11	50	56	5	35	32	0	1,5	32
12	71	54	5	49	46	0	0,5	46
13	304	18	5	270	270	5	5	258
14	365	13	5	334	334	3	5	323
15	560	1	5	560	595	7	5	556
16	21	91	5	12	12	9	5	11
17	32	78	5	20	20	11	5	18
18	129	14	5	122	122	8	5	113
19	163	12	5	158	158	8	5	146
20	268	0	5	268	280	5	5	267

Ici aussi, les résultats de la recherche locale sont meilleures.

(c) Classe d'instances D

Instances	AG_VAL	POURC_OPT	AG_T	MIN(PCC,ACM)	RL_VAL	POURC_OPT	RL_T	OPT
1	107	1	5	107	112	6	5	106
2	231	5	5	237	259	18	5	220
3	1599	2	5	1605	1665	6	5	1565
4	1972	2	5	1972	2045	6	5	1935
5	3347	3	5	3266	3314	2	5	3250
6	71	6	5	73	75	12	5	67
7	105	2	5	105	121	17	5	103
8	1122	5	5	1122	1186	11	5	1072
9	1493	3	5	1510	1577	9	5	1448
10	2198	4	5	2132	2156	2	5	2110
11	30	3	5	32	35	21	5	29
12	42	0	5	43	52	24	5	42
13	522	4	5	522	545	9	5	500
14	683	2	5	683	731	10	5	667
15	1165	4	5	1135	1145	3	5	1116
16	16	23	5	16	18	38	5	13
17	25	9	5	25	41	78	5	23
18	241	8	5	241	264	18	5	223
19	330	6	5	330	339	9	5	310
20	555	3	5	543	540	1	5	537

Ici, ce sont les résultats issues d'une évolution suivant l'algorithme génétique qui sont les meilleures. Comme explication, nous pouvons dire que du fait de la taille des instances, très peu d'améliorations sont trouvées avec la recherche locale qui passe en outre beaucoup de temps à rechercher un voisin améliorant.

(d) Classe d'instances E

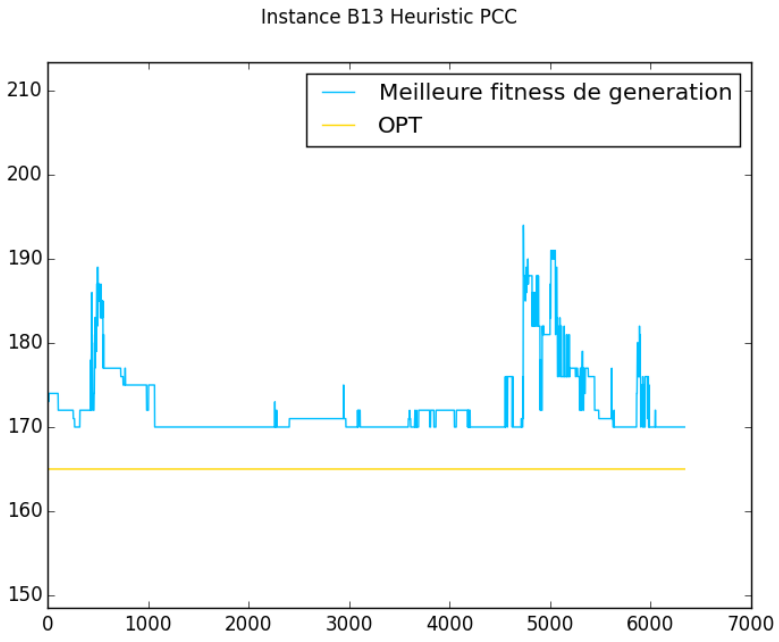
	AG_VAL	POURC_OPT	AG_T	MIN(PCC, ACM)	RL_VAL	POUR_OPT	RL_T	OPT
1	131	18	5	125	124	12	5	111
2	335	57	5	255	331	55	5	214
3	4473	11	5	4181	4442	11	5	4013
4	5548	9	5	5207	5561	9	5	5101
5	8335	3	5	8168	8327	2	5	8128
6	157	115	5	86	109	49	5	73
7	245	69	5	169	191	32	5	145
8	3115	18	5	2751	3096	17	5	2640
9	4051	12	5	3715	4020	12	5	3604
10	5911	6	5	5643	5913	6	5	5600
11	52	53	5	39	44	29	5	34
12	125	87	5	71	102	52	5	67
13	1488	16	5	1345	1481	16	5	1280
14	1937	12	5	1784	1926	11	5	1732
15	2961	6	5	2818	2950	6	5	2784
16	48	220	5	18	34	127	5	15
17	82	228	5	29	71	184	5	25
18	646	15	5	615	634	12	5	564
19	856	13	5	791	839	11	5	758
20	1393	4	5	1355	1381	3	5	1342

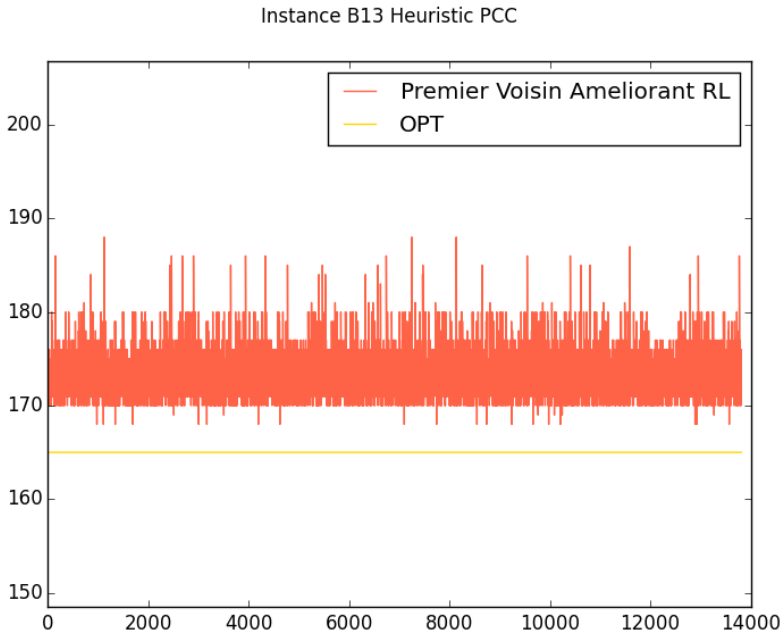
Les conclusions semblent s'inverser ici où la recherche locale retourne de meilleures résultats. En effet, le temps de calcul des individus issus de la randomisation devient important, la génération des individus de la population de base dure beaucoup plus longtemps et laisse peu de temps à l'exécution de l'algorithme génétique.

Des graphiques montrant l'évolution au cours du temps de l'application de ces métaheuristiques peuvent nous en apprendre plus. C'est l'objet du paragraphe suivant.

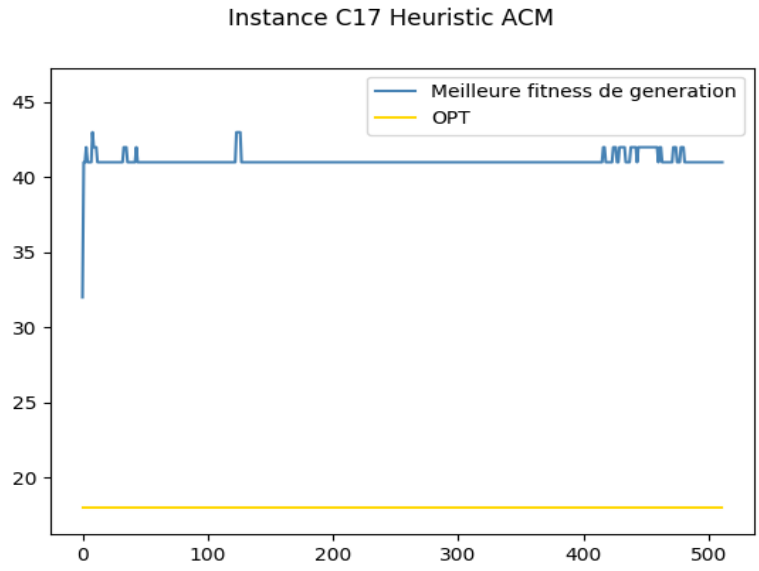
4. Graphiques d'évolution

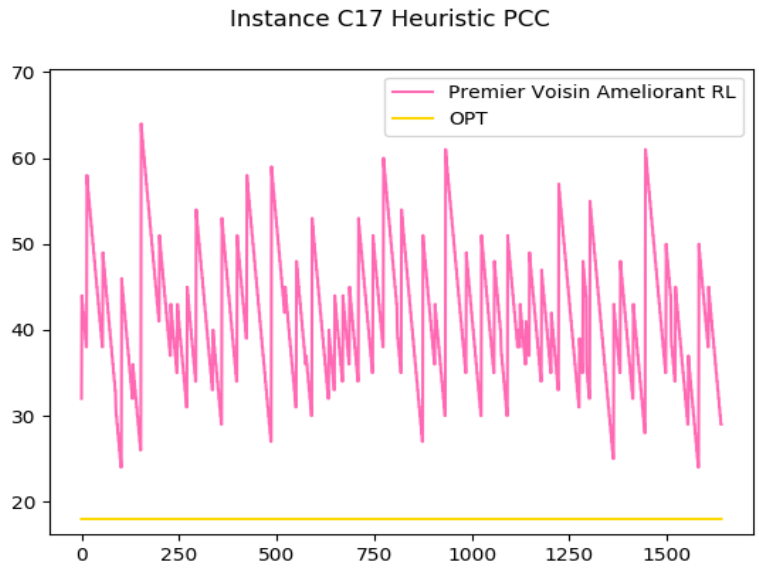
✔ Instance B13



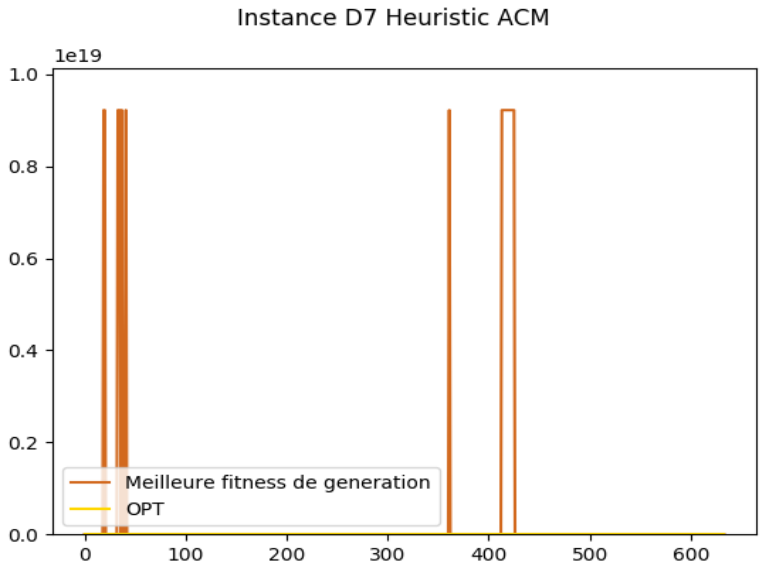


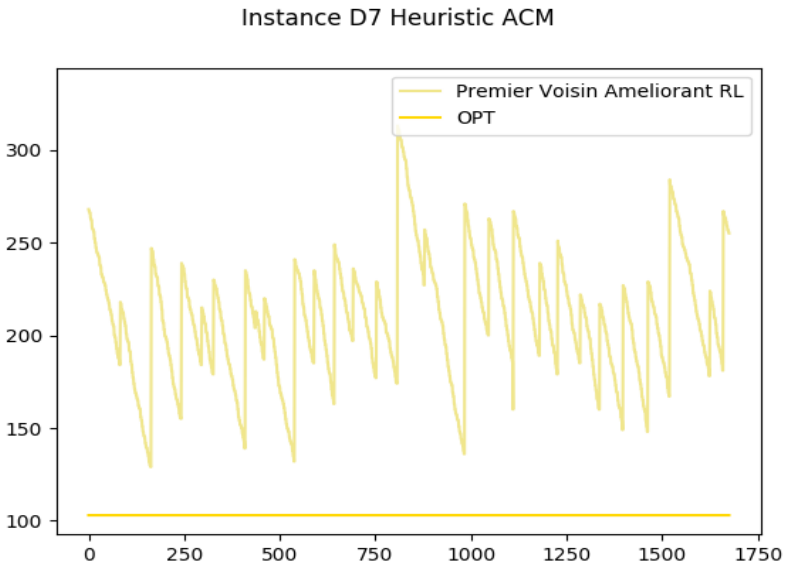
✔ Instance C17



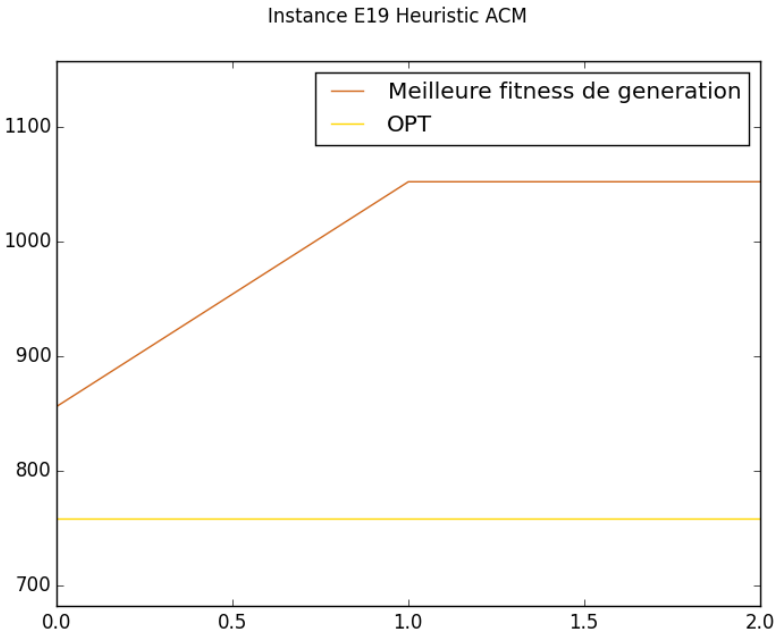

Instance C17 Heuristic PCC
Premier Voisin Ameliorant RL
OPT
70
60
50
40
30
20
0
250
500
750
1000
1250
1500

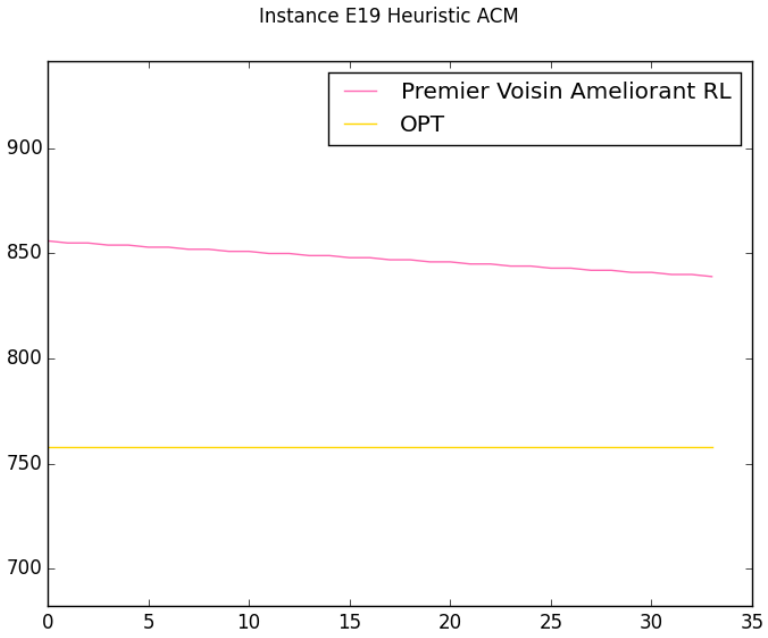
✔ Instance D7


Instance D7 Heuristic ACM
1e19
1.0
0.8
0.6
0.4
0.2
0.0
Meilleure fitness de generation
OPT
0
100
200
300
400
500
600



✔ Instance E19





Conclusion

En conclusion, nous pouvons dire que ce projet a été pour nous très enrichissant d'une part parce qu'il nous a permis d'appliquer certaines métaheuristiques qui nous ont été enseignées en cours sur un problème assez célèbre qu'on ne connaissait pas. Pour ce type de problème (et en fonction des contraintes qui nous ont été imposéees (timeout)), la métaheuristique de recherche locale nous semble la plus efficace.

Il nous a en outre donné l'occasion de nous servir d'une librairie libre de gestion de graphes : PyAlgDat.