Sorbonne université

Rapport de RP
Résolution de Problèmes

Etudiants :

Baptiste Jarry
Thomas Zanivan

Chargé de TD :

Thibaut Lust

Année 2017-2018

1 Introduction

Nous avons fait le choix d'implémenter notre projet en C afin de tirer parti de la rapidité d'éxécution de ce langage. Nous n'avons pas utilisé de librairie de graphe mais réimplémentons notre propre structure de graphe (décrit dans le fichier types.h).
Dans notre code, un graphe est représenté par un nombre de noeuds, un nombre d'arêtes, un nombre de noeux terminaux et non-terminaux, ainsi que des tableaux de noeuds, d'arêtes, de noeux terminaux et non-terminaux.
Les fonctions de traitement principales se trouvent dans le fichier graphe.c, quelques fonctions annexes se trouvent également dans le fichier utils.c.

2 Algorithme Génétique

Comme indiqué dans le sujet, nous utilisons pour l'ensemble du projet un codage des individus ne comportant que les noeuds non-terminaux avec des valeurs binaires pour indiquer si le noeud est présent dans l'arbre de Steiner.
De même, nous avons implémenté l'algorithme avec un opérateur à un point qui copie une partie de taille aléatoire d'un parent d'un coté et de l'autre parent de l'autre coté.

2.1 Remplacement et nouvelle génération

L'implémentation du code a été faite uniquement avec un remplacement élitiste. En effet, le remplacement générationnel est problématique car il détruit des solutions parfois meilleures que les croisements ; cela dégrade donc souvent la qualité de la population au lieu de l'améliorer, et une évolution ponctuelle risque de ne pas être conservée.

Les 3 figures ci-dessous représentent le même graphique avec des niveaux de zoom différents pour mieux visualiser ce qui se passe à différentes échelles.

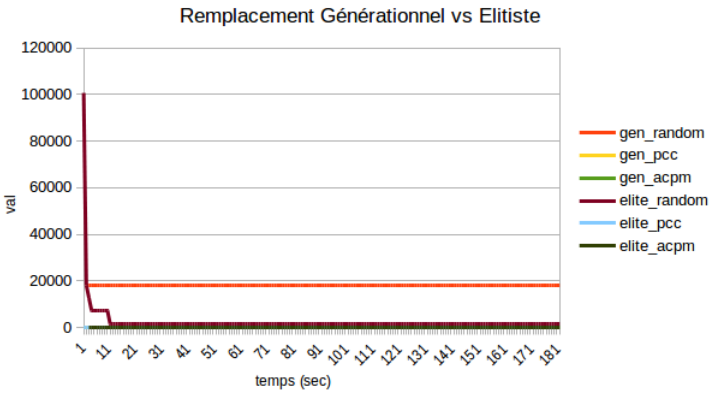

FIGURE 1 – vision d'ensemble

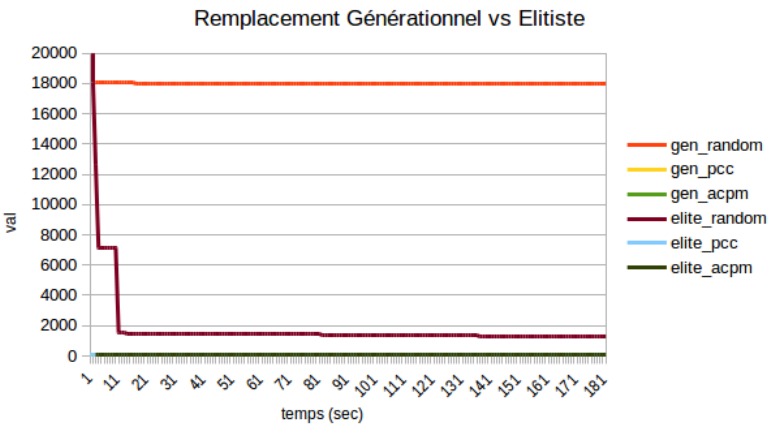

FIGURE 2 – premier zoom

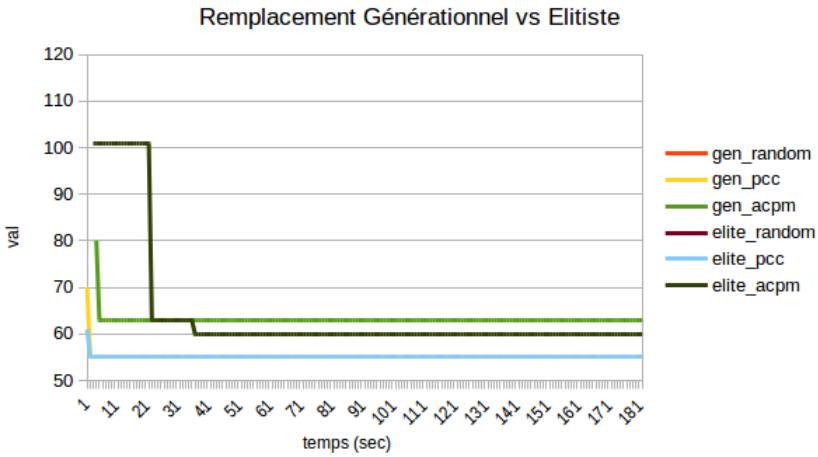

FIGURE 3 – second zoom

On observe que elite-random est meilleur que gen-random (1er et 2nd graphique), que elite-pcc est légèrement meilleur que gen-pcc (3e graphique) et que elite-acpm est meilleur que gen-acpm (3e graphique).

Nous avons donc logiquement fait le choix de continuer avec la stratégie de remplacement élitiste puisqu'elle donne les meilleurs résultats quelle que soit l'heuristique.

2.2 Population initiale

Au tout début du développement de l'algorithme génétique, l'algorithme démarrait avec une population nulle (aucun noeud, pour chaque individu), et c'étaient alors les mutations aléatoires qui permettaient d'avoir une meilleure solution. L'ajout d'une population générée aléatoirement donne une très grande diversité des solutions et permet un démarrage plus rapide.

Cependant, une génération purement aléatoire de la population initiale donne une population de faible qualité et donc une évolution assez lente. Cela est d'autant plus visible sur des instances un peu complexes (par exemple on l'observe bien sur les groupes C,D,E).

L'ajout d'heuristiques pour générer la population permet d'avoir des individus plus intéressants car d'une part ils sont viables (la solution est réalisable, il s'agit bien d'arbres de Steiner) et d'autre part ce sont de bonnes approximations de la solution optimale. On a donc une population qui, dès la première itération, contient des individus de bonne qualité qui ont plus de chances d'améliorer la solution globale.

2.3 Analyse de la taille de la population

Nous avons utilisé comme valeur de référence une population de taille 100 pour nos tests. Nous avons également testé l'algorithme génétique sur une population de 50 individus et sur une population de 500 individus comme on peut l'observer sur les deux graphiques suivants.

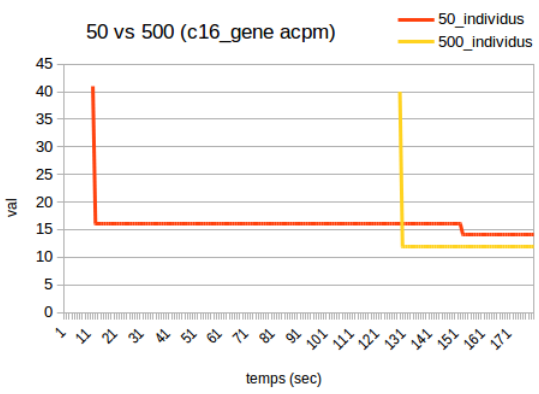

FIGURE 4 – c16, opt=11

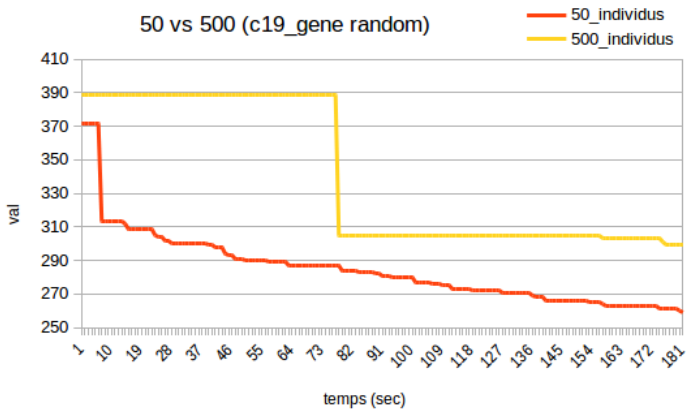

FIGURE 5 – c19 génétique random, opt=146

Remarque : l'absence de valeur observée au début du graphique de gauche s'explique ici par le fait qu'on visualise une instance d'assez grande taille, donc l'heuristique met un certain temps à obtenir une population initiale, ce qui est d'autant plus long que la population est grande.

On a pu observer sur une grande partie des instances de petite taille ou de taille moyenne qu'une population de 500 individus met plus longtemps à converger vers la solution optimale qu'une population de 50 individus. Cela s'explique par le fait qu'avec moins d'invidus (mais assez pour avoir une population diverse), la solution évolue plus rapidement car on peut obtenir plus de générations.

Cependant, dans certains cas d'instances complexes - comme sur le graphique de gauche - on observe qu'une fois la population de 500 individus générée, elle trouvera potentiellement une meilleure solution, car sa population est plus diversifiée.

3 Heuristiques de construction

Dans les deux heuristiques suivantes, nous avons implémenté une randomisation des poids des arètes du graphe entre -20% et +20%. Cette randomisation s'effectue pour chaque arète avant de générer un individu afin d'obtenir une population initiale variée.

3.1 Heuristique du plus court chemin (PCC)

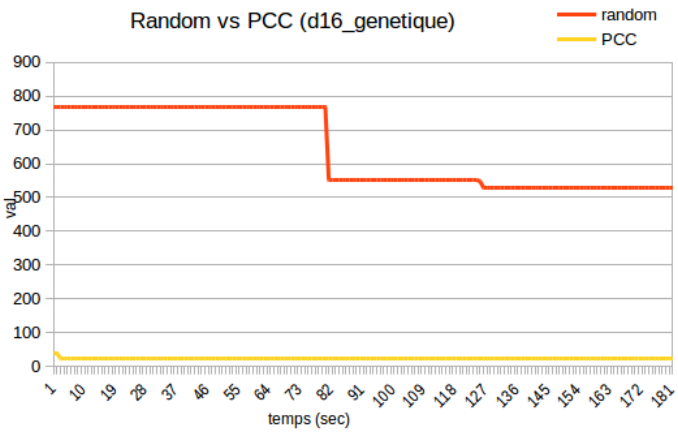

FIGURE 6 – opt=13 et pcc converge en 23

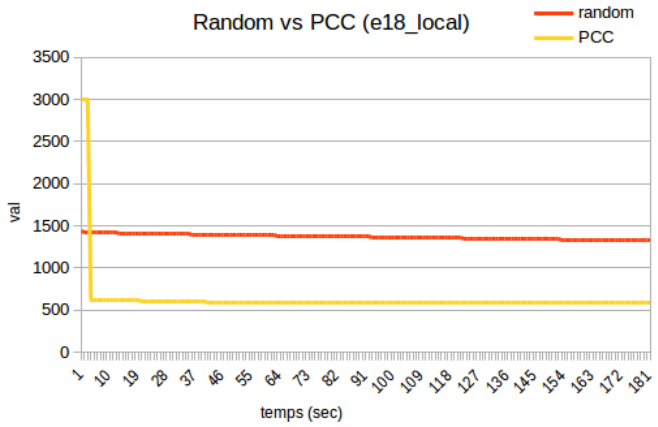

FIGURE 7 – opt=564 et pcc converge en 589

On peut observer que l'heuristique de plus court chemin permet une solution initiale bien meilleure et donc une convergence vers une solution proche de l'optimum bien plus rapide. Dans certains cas la solution optimale n'est pas meilleure mais la convergence est quand même beaucoup plus rapide.

3.2 Heuristique de l'arbre couvrant minimum (ACPM)

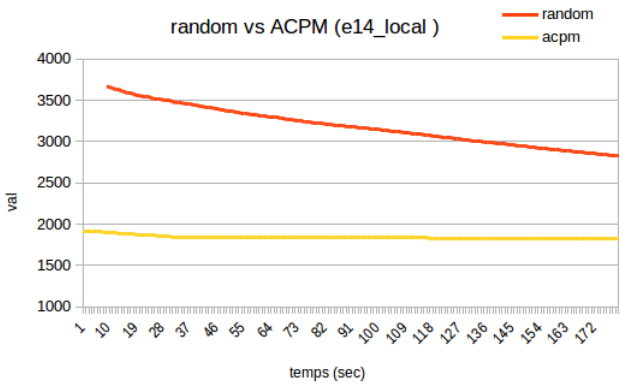

FIGURE 8 – e14 local, opt=1732

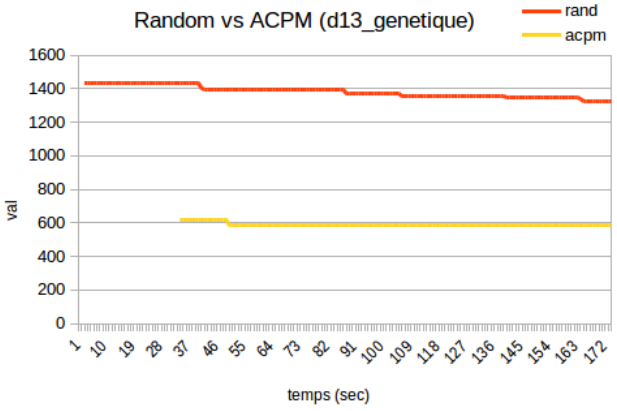

FIGURE 9 – d13 génétique, opt=500

Sans surprise, on observe bien que comme pour l'heuristique PCC, l'heuristique ACPM donne une meilleure solution initiale que l'heuristique aléatoire et permet donc une convergence plus rapide vers l'optimum.

3.3 Comparaison entre l'heuristique PCC vs ACPM

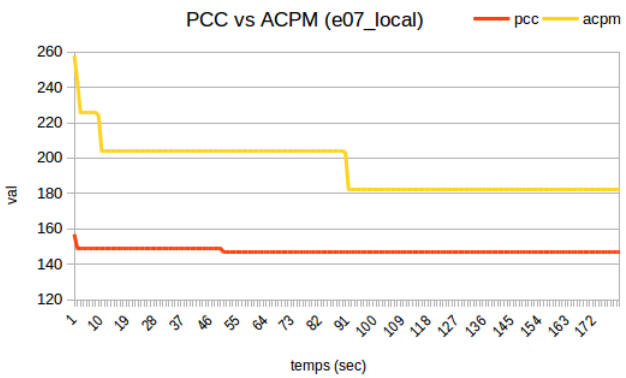

FIGURE 10 – e07 local, opt = 145

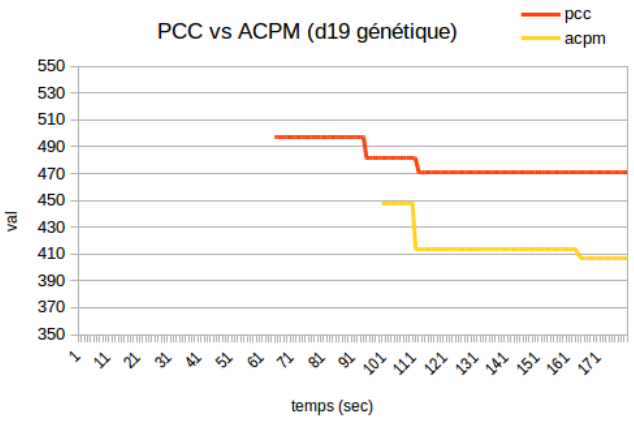

FIGURE 11 – d19 génétique, opt=310

Sur le graphique de gauche on voit que l'heuristique PCC est meilleure que l'heuristique ACPM, ce que l'on a pu confirmer sur la quasi-totalité des instances, notamment avec l'algorithme de recherche locale (cf tableaux compilant toutes les données à la fin du rapport).
Cependant, pour certains cas complexes avec l'algorithme génétique, l'heuristique ACPM s'avère donner un résultat légèrement meilleur, comme on peut l'observer sur le graphique de droite. Ces cas restent cependant minoritaires.

4 Recherche locale

Dans le cas où l'algorithme de recherche locale dispose d'assez de temps pour être relancé, nous avons ajouté une fonction de randomisation des poids des arêtes du graphe lors de chaque nouveau relancement, afin d'obtenir des solutions plus variées et garder la meilleure.

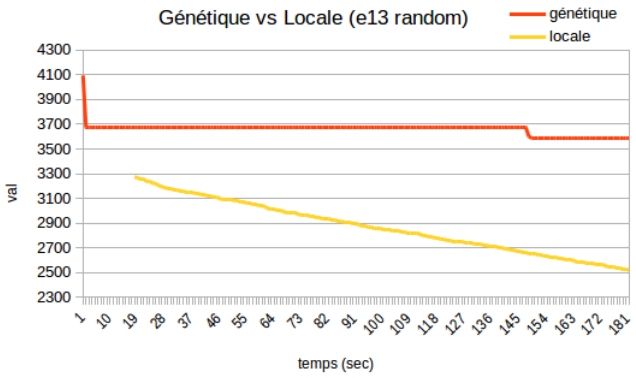

FIGURE 12 – e13 random

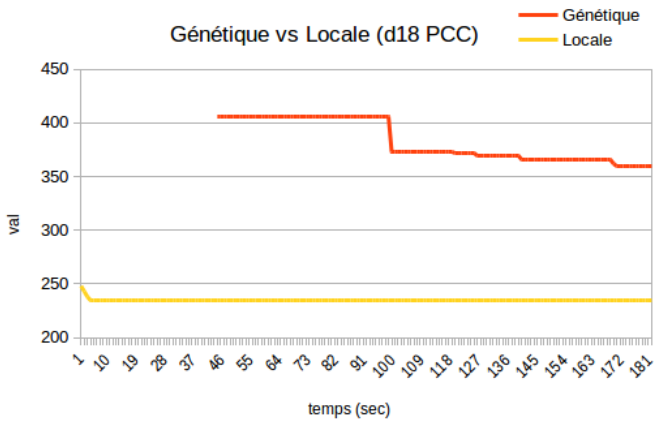

FIGURE 13 – d18 pcc

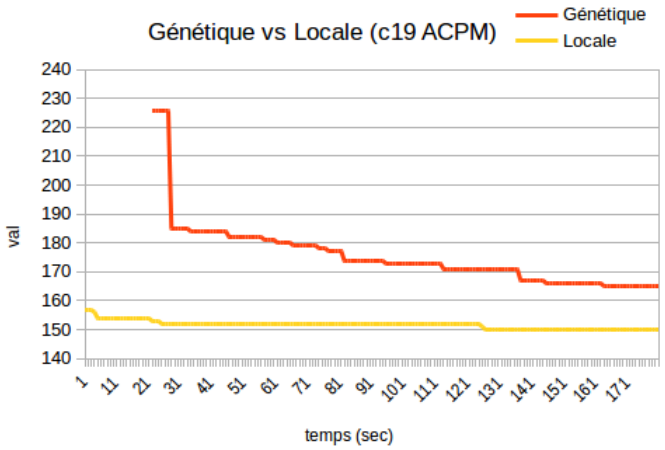

FIGURE 14 – c19 acpm

Comme on le voit sur les trois graphiques ci-dessus, on a pu observer sur la majorité des instances que la recherche locale est plus efficace que l'algorithme génétique, peu importe l'heuristique utilisée.

5 Instances et évaluation

Nous avons testé chaque instance avec un temps de 3 minutes pour chaque méthode et heuristique, soit 18 minutes par instance. Un fichier (contenant les temps et la valeur actuelle de la solution) correspond à chacun de ces tests dans le dossier Output. Les résultats sont compilés dans les tableaux des deux dernières pages.

6 Conclusion

A travers l'implémentation et l'analyse de ces deux algorithmes et de ces trois méthodes heuristiques d'initiation des données, nous avons pu constater que la combinaison qui converge le plus proche de l'optimal en 3 minutes est l'algorithme de recherche locale avec l'heuristique de plus court chemin. Sur les instances du groupe E, on observe d'ailleurs que cette combinaison obtient un pire cas de 27% d'écart à l'optimum (19 au lieu de 15) pour l'instance e16 et que c'est le meilleur score de toutes les combinaisons testées. Sur le groupe E, la combinaison *recherche locale+PCC* obtient une moyenne d'écart à l'optimum de seulement 3%. Ensuite viennent *recherche locale+ACPM* avec une moyenne de 20% puis *génétique+PCC* avec 23% et enfin *génétique+ACPM* avec 36%, les autres obtenant une moyenne d'écart à l'optimal extrêmement élevée.

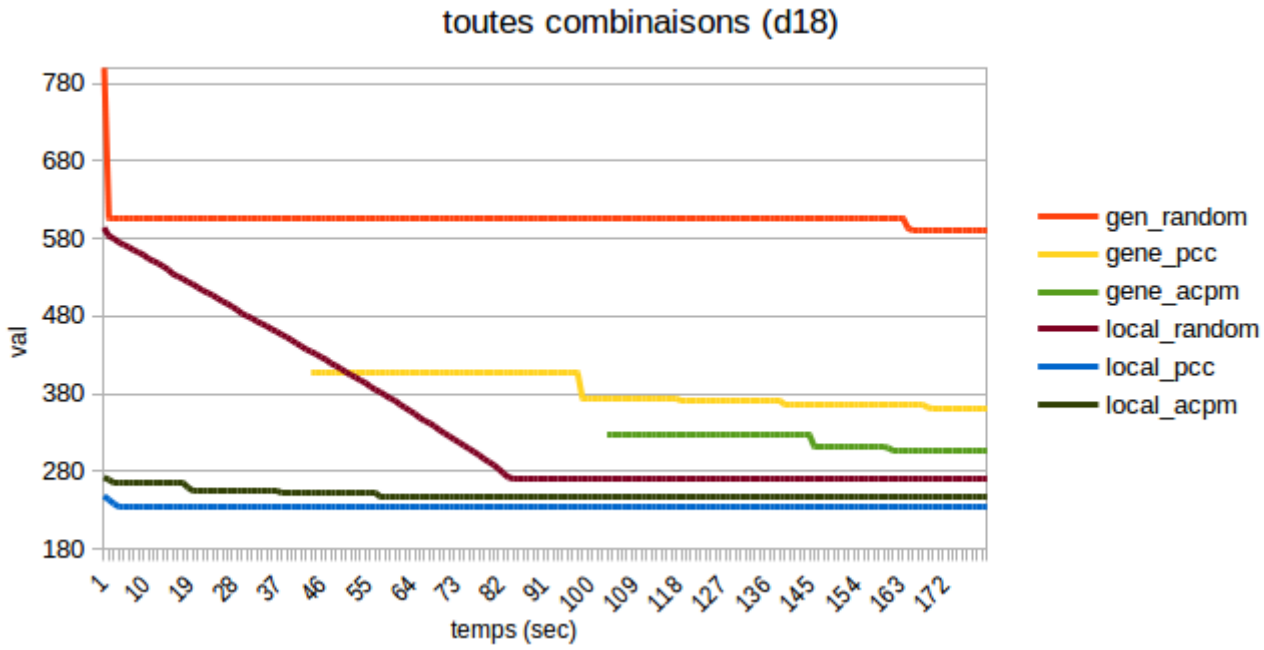

FIGURE 15 – d18, toutes combinaisons testées

L'algorithme génétique donne aussi d'assez bons résultats, mais converge moins rapidement que la recherche locale. De plus, la génération des heuristiques sur les grosses instances (4-5 dernières de chaque groupe) est lente et l'algorithme génétique n'a même pas le temps de démarrer par la suite. C'est ce qui explique les valeurs assez grandes des écarts à l'optimum pour ces instances : il s'agit en fait de la solution donnée par l'heuristique, parfois bien après la limite de 3 minutes fixée au départ.

TABLE 1 – Scores des différents algorithmes, les % sont les écarts à l'opt

Inst	opt	gene rand score	%écart	gene pcc score	%	gene acpm score	%	local rand score	%	local pcc score	%	local acpm score	%
b01	82	82	0%	82	0%	82	0%	82	0%	82	0%	82	0%
b02	83	83	0%	83	0%	83	0%	83	0%	83	0%	83	0%
b03	138	138	0%	138	0%	138	0%	138	0%	138	0%	138	0%
b04	59	59	0%	59	0%	59	0%	59	0%	59	0%	59	0%
b05	61	61	0%	61	0%	61	0%	61	0%	61	0%	61	0%
b06	122	124	2%	124	2%	122	0%	122	0%	122	0%	122	0%
b07	111	111	0%	111	0%	112	1%	111	0%	111	0%	111	0%
b08	104	107	3%	104	0%	107	3%	104	0%	104	0%	104	0%
b09	220	220	0%	220	0%	220	0%	220	0%	220	0%	220	0%
b10	86	86	0%	86	0%	86	0%	86	0%	86	0%	86	0%
b11	88	101	15%	90	2%	91	3%	90	2%	88	0%	88	0%
b12	174	174	0%	174	0%	174	0%	174	0%	174	0%	174	0%
b13	165	170	3%	170	3%	170	3%	165	0%	165	0%	168	2%
b14	235	240	2%	235	0%	239	2%	235	0%	235	0%	240	2%
b15	318	320	1%	318	0%	320	1%	320	1%	318	0%	321	1%
b16	127	136	7%	127	0%	141	11%	127	0%	127	0%	132	4%
b17	131	133	2%	131	0%	132	1%	131	0%	131	0%	132	1%
b18	218	218	0%	218	0%	218	0%	219	0%	218	0%	218	0%
c01	85	1565	1741%	85	0%	102	20%	86	1%	85	0%	85	0%
c02	144	5024	3389%	144	0%	163	13%	148	3%	144	0%	152	6%
c03	754	1449	92%	757	0%	788	5%	765	1%	755	0%	765	1%
c04	1079	1534	42%	1090	1%	1128	5%	1100	2%	1083	0%	1086	1%
c05	1579	1638	4%	1579	0%	1587	1%	1580	0%	1579	0%	1587	1%
c06	55	1261	2193%	55	0%	60	9%	60	9%	55	0%	57	4%
c07	102	1204	1080%	102	0%	137	34%	114	12%	102	0%	109	7%
c08	509	1172	130%	512	1%	534	5%	546	7%	510	0%	524	3%
c09	707	1190	68%	708	0%	744	5%	724	2%	708	0%	725	3%
c10	1093	1204	10%	1093	0%	1100	1%	1095	0%	1093	0%	1096	0%
c11	32	500	1463%	33	3%	42	31%	40	25%	32	0%	36	13%
c12	46	505	998%	47	2%	47	2%	56	22%	46	0%	46	0%
c13	258	542	110%	264	2%	268	4%	266	3%	260	1%	265	3%
c14	323	539	67%	331	2%	335	4%	332	3%	326	1%	330	2%
c15	556	640	15%	559	1%	563	1%	559	1%	556	0%	562	1%
c16	11	221	1909%	18	64%	14	27%	13	18%	12	9%	12	9%
c17	18	226	1156%	24	33%	28	56%	25	39%	20	11%	21	17%
c18	113	267	136%	145	28%	124	10%	122	8%	117	4%	118	4%
c19	146	272	86%	182	25%	165	13%	152	4%	150	3%	150	3%
c20	267	331	24%	312	17%	281	5%	268	0%	267	0%	267	0%

TABLE 2 – Scores des différents algorithmes, les % sont les écarts à l'opt

Inst	opt	gene rand score	%écart	gene pcc score	%	gene acpm score	%	local rand score	%	local pcc score	%	local acpm score	%
d01	106	181K	170K%	107	1%	131	24%	119	12%	106	0%	108	2%
d02	220	140K	63K%	229	4%	281	28%	249	13%	224	2%	246	12%
d03	1565	142K	8958%	1590	2%	1735	11%	2K	5%	1570	0%	1603	2%
d04	1935	83515	4216%	1950	1%	2056	6%	2K	2%	1941	0%	1971	2%
d05	3250	11375	250%	3253	0%	3296	1%	1003K	31K%	3251	0%	3261	0%
d06	67	13987	21K%	70	4%	87	30%	87	30%	67	0%	71	6%
d07	103	3063	2874%	103	0%	185	80%	145	41%	103	0%	115	12%
d08	1072	3158	195%	1107	3%	1227	14%	1K	6%	1080	1%	1134	6%
d09	1448	3125	116%	1474	2%	1574	9%	2K	4%	1453	0%	1486	3%
d10	2110	2955	40%	2122	1%	2156	2%	2K	1%	2110	0%	2121	1%
d11	29	1325	4469%	30	3%	36	24%	41	41%	29	0%	34	17%
d12	42	1262	2905%	42	0%	55	31%	73	74%	42	0%	46	10%
d13	500	1325	165%	529	6%	586	17%	542	8%	510	2%	529	6%
d14	667	1345	102%	692	4%	746	12%	697	4%	672	1%	692	4%
d15	1116	1440	29%	1143	2%	1146	3%	1K	1%	1119	0%	1124	1%
d16	13	530	3977%	23	77%	17	31%	37	185%	15	15%	18	38%
d17	23	524	2178%	40	74%	50	117%	64	178%	26	13%	37	61%
d18	223	591	165%	360	61%	307	38%	271	22%	235	5%	247	11%
d19	310	627	102%	471	52%	407	31%	340	10%	324	5%	328	6%
d20	537	751	40%	685	28%	601	12%	538	0%	539	0%	540	1%
e01	111	2867K	2582K%	113	2%	136	23%	199	79%	111	0%	120	8%
e02	214	2791K	1304K%	234	9%	364	70%	305	43%	227	6%	305	43%
e03	4013	2315K	58K%	4139	3%	4448	11%	1005K	25K%	4048	1%	4173	4%
e04	5101	1797K	35K%	5188	2%	5489	8%	3005K	59K%	5133	1%	5231	3%
e05	8128	1098K	13K%	8182	1%	8342	3%	7008K	86K%	8143	0%	8164	0%
e06	73	775K	1061K%	78	7%	138	89%	4K	5K%	76	4%	94	29%
e07	145	751K	518K%	150	3%	203	40%	4K	2K%	147	1%	182	26%
e08	2640	639K	24K%	2746	4%	3049	15%	3K	7%	2683	2%	2797	6%
e09	3604	566K	16K%	3717	3%	4058	13%	1004K	28K%	3643	1%	3754	4%
e10	5600	230K	4001%	5655	1%	5892	5%	1006K	18K%	5610	0%	5654	1%
e11	34	3713	11K%	34	0%	40	18%	3K	8K%	34	0%	34	0%
e12	67	3612	5291%	68	1%	89	33%	3K	4K%	67	0%	80	19%
e13	1280	3587	180%	1387	8%	1506	18%	3K	97%	1310	2%	1380	8%
e14	1732	3811	120%	1841	6%	1973	14%	3K	63%	1758	2%	1827	5%
e15	2784	3904	40%	2870	3%	2930	5%	3K	2%	2793	0%	2816	1%
e16	15	1390	9167%	28	87%	31	107%	2K	11K%	19	27%	32	113%
e17	25	1360	5340%	66	164%	64	156%	2K	6K%	27	8%	54	116%
e18	564	1514	168%	979	74%	827	47%	1K	135%	589	4%	650	15%
e19	758	1616	113%	1196	58%	1006	33%	2K	123%	781	3%	801	6%
e20	1342	1906	42%	1775	32%	1547	15%	2K	21%	1349	1%	1353	1%