Sorbonne Université
Faculté des Sciences et Ingénierie

Master d'informatique

4I201 - Résolution de problèmes

Rapport du projet : Métaheuristiques pour la résolution du problème de l'arbre de Steiner de poids minimum

B. Thanh Luong, 3504859
Hans Thirunavukarasu, 3605592

Encadrant :
M. Thibaut Lust

mai 2017

Table des matières

1 Introduction

Le problème de l'arbre de Steiner de poids minimum est NP-difficile. Étant donné un graphe connexe G = (V, E) et un ensemble T de sommets dits terminaux, nous allons chercher un ensemble de sommets non terminaux qui relient les sommets terminaux créant un arbre couvrant de poids minimum.
L'objectif de ce projet est d'utiliser 3 approches qui résolvent le problème en temps polynomial : un algorithme génétique, une heuristique de construction et une recherche locale.
L'algorithme génétique a été conçu de manière générale et aléatoire afin d'avoir une population variée d'individus différents. Grâce à la variation de la population et aux différentes méthodes de génération d'une nouvelle population à chaque étape, la convergence vers un optimum local est rapide.
Nous allons utiliser deux heuristiques qui sont le plus court chemin et l'arbre couvrant minimum pour améliorer l'approche du problème.
Enfin, la recherche locale sera utilisé pour avoir une perspective globale de problème. Nous testerons nos algorithmes sur des instances des ensembles B, C, D, et E disponibles à http ://steinlib.zib.de/testset.php.

2 Algorithme génétique

2.1 Population initiale

N'importe quel individu (réalisable ou non réalisable) est codé par un vecteur binaire pour chaque sommet non-terminal, prenant 1 si le sommet est présent dans le graphe (ou la forêt dans le cas non réalisable) et 0 sinon.
La première phase consiste à construire une population aléatoire dont les individus ayant chaque bit une probabilité entre 0.2 et 0.5 d'être pris.
Ensuite, nous allons construire des générations suivantes à partir de la population initiale. Nous proposons 2 types de sélection des parents de la population courante ainsi que 2 stratégies de remplacement de population.
Sélection des parents : Parmi les individus, le moyen simple est de choisir celui qui a le meilleur score. Nous proposons une deuxième façon : pour chaque individu, nous le prenons comme parent si un tirage aléatoire est inférieur au ratio $\frac{best\ score}{son\ score}$.
Production des enfants : S'il y a un unique parent choisi, pour chaque bit, une probabilité entre 0.01 et 0.04 de le changer de 0 à 1 et inversement. Sinon nous faisons un opérateur de croisement, pour chaque pair de parents, nous utilisons le croisement au milieu pour reproduire deux enfants.
Remplacement de population : Le remplacement générationnel consiste à écraser complètement la population courante par la nouvelle. Le remplacement élitiste est la sélection des meilleurs individus parmi les parents et les enfants avec une probabilité comme la sélection des parents.

Après avoir testé toutes les combinaisons des processus possibles, nous constatons que la convergence vers le minimum des croisements est très lente et aussi loin de l'optimum des problèmes. Nous allons donc rajouter dans la population initiale 2 individus spéciales qui sont les résultats des heuristiques du plus court chemin et de l'arbre couvrant minimum. Ces 2 individus ont pour but d'accélérer la convergence.

2.2 Heuristique du plus court chemin

Cette heuristique donne une des meilleurs approches dans la population initiale. Elle consiste à la construction d'un graphe complet des sommets terminaux. Si deux sommets terminaux ne sont pas connecter le coût de l'arête connectant deux sommets est égale au plus court chemin entre ces deux dans le graphe initiale. Une fois le graphe complet fait, nous remplaçons toutes les arêtes "virtuelles" par le chemin réel, puis construisons un arbre couvrant minimum tous les sommets du graphe et enfin enlevons toutes les feuilles redondantes (i.e. les sommets non-terminaux de degré 1).
Cette heuristique est 2 - $\frac{2}{|T|}$ - OPT.

2.3 Heuristique de l'arbre couvrant minimum

Cette heuristique est moins efficace que celle du plus court chemin mais donne aussi une solution assez proche de l'optimum. Nous construisons l'arbre couvrant minimum du problème, puis la récursions est faite jusqu'à ce que l'on ne puisse plus enlever les feuilles redondantes.

2.4 Diversification la population initiale par randomisation des heuristiques de construction

La population aléatoire n'est pas efficace pour converger vers l'optimum. On converge très souvent vers celui du plus court chemin.
De ce fait, les individus aléatoires seront remplacés par ceux des deux heuristiques proposées en perturbant le poids des arêtes. C'est-à-dire, une nouvelle instance de l'arbre sera crée en changeant le poids associé aux arêtes de -0.2 à -0.05 ou de 0.05 à 0.2. Quand nous obtenons le résultat des heuristiques, nous reviendrons vers le problème initial des nouvelles instances.

3 Recherche locale

En partant de la population initiale générée par les heuristiques et la randomisation des heuristiques des constructions, nous allons faire une recherche locale suivante :

Algorithme 1 : Recherche locale

Données : I : Meilleur individu de la population initiale
Résultat : Résultat de la recherche locale

```
faire
    Voisinage ← ∅
    pour noeud N ∈ non-terminaux faire
        si N prise dans I alors
            si degré(N) = 1 alors
                Nouveau voisin = I avec N non pris
                si Nouveau voisin a un meilleur score que I alors
                    si Voisinage ≠ ∅ alors
                        si Nouveau voisin a un meilleur score que celui dans le voisinage
                         alors
                            Voisinage = {Nouveau voisin}
                        fin
                    sinon
                        Voisinage = {Nouveau voisin}
                    fin
                fin
            sinon
                Nouveau voisin = I avec N non pris
                si Nouveau voisin a un meilleur score que I alors
                    si Voisinage ≠ ∅ alors
                        si Nouveau voisin a un meilleur score que celui dans le voisinage
                         alors
                            Voisinage = {Nouveau voisin}
                        fin
                    sinon
                        Voisinage = {Nouveau voisin}
                    fin
                fin
            fin
        sinon
            Nouveau voisin = I avec N pris
            si Nouveau voisin a un meilleur score que I alors
                si Voisinage ≠ ∅ alors
                    si Nouveau voisin a un meilleur score que celui dans le voisinage alors
                        Voisinage = {Nouveau voisin}
                    fin
                sinon
                    Voisinage = {Nouveau voisin}
                fin
            fin
        fin
    fin
    si Il y a un voisin dans la voisinage alors
        I ← Le voisin
        Non convergence
    sinon
        retourner I
    fin
tant que Non convergence;
```

4 Analyse des résultats

Pour le premier testset B, nous avons testé toutes les combinaisons de sélections de parents avec remplacement de population. Les sélections sont BP (les individus avec le meilleur score dans la population) et MP (les individus ayant les meilleurs score, choisis par la probabilité expliqué dans la section 2.1). Les remplacements sont E (élitiste) et G (générationnel). Pour chaque combinaison nous avons fait une boucle de 10 itérations afin d'avoir des moyenne plus précise. Partant de la population initiale avec 2 heuristiques du PCC (plus court chemin) et CM (arbre couvrant minimum) ainsi que des individus aléatoires, nous constatons que pour les petites instances de ce testset, MP-E et MP-G donnent des meilleurs rapports.
Cependant, ces deux stratégies donnent un temps d'exécution important donc nous avons décidé pour la suite d'utilisé BP-E pour tester la population initiale avec des individus randomisés sur le poids(RAN).
La recherche locale (LOC) est basée sur la population initiale randomisée sur le poids.
Voici les résultats du testset B de notre algorithme :

Instance - OPT	Algorithme	Résultat	Ratio	Variance	Écart-type	Temps d'exécution
B/b01.stp - 82	PCC	82	1			0.00451
	CM	83	1.012			0.00087
	BP-E	82.0	1	0.0	0.0	0.14385
	BP-G	82.0	1	0.0	0.0	0.53851
	MP-E	82.0	1	0.0	0.0	1.34391
	MP-G	82.0	1	0.0	0.0	1.25432
	RAN	82	1			0.28891
	LOC	82	1			0.31425
B/b02.stp - 83	PCC	90	1.084			0.00602
	CM	96	1.157			0.00102
	BP-E	89.0	1.072	0.0	0.0	0.12763
	BP-G	88.9	1.071	0.09	0.3	2.05193
	MP-E	88.4	1.065	0.64	0.8	1.04937
	MP-G	88.2	1.063	3.76	1.93907	1.54754
	RAN	83	1			0.95667
	LOC	83	1			0.49511
B/b03.stp - 138	PCC	140	1.014			0.0236
	CM	144	1.043			0.00074
	BP-E	138.8	1.006	0.16	0.4	1.00194
	BP-G	138.9	1.007	0.09	0.3	0.57525
	MP-E	138.0	1	0.0	0.0	1.4193
	MP-G	138.0	1	0.0	0.0	1.49279
	RAN	138	1			0.34568
	LOC	138	1			0.42526
B/b04.stp - 59	PCC	62	1.051			0.00575
	CM	63	1.068			0.00164
	BP-E	61.7	1.046	0.81	0.9	0.79119
	BP-G	62.0	1.051	0.0	0.0	0.16545
	MP-E	59.1	1.002	0.09	0.3	1.39856
	MP-G	59.6	1.010	0.84	0.91652	1.51993
	RAN	59	1			0.47931
	LOC	59	1			0.4395

Instance - OPT	Algorithme	Résultat	Ratio	Variance	Écart-type	Temps d'exécution
B/b05.stp - 61	PCC	64	1.049			0.00806
	CM	68	1.115			0.00116
	BP-E	61.0	1	0.0	0.0	1.28889
	BP-G	61.0	1	0.0	0.0	1.2023
	MP-E	61.0	1	0.0	0.0	1.863
	MP-G	61.0	1	0.0	0.0	1.84786
	RAN	61	1			0.73086
	LOC	61	1			0.5936
B/b06.stp - 122	PCC	128	1.049			0.03117
	CM	133	1.090			0.00091
	BP-E	124.0	1.016	0.0	0.0	0.38745
	BP-G	124.0	1.016	0.0	0.0	0.2602
	MP-E	124.0	1.016	0.0	0.0	1.60832
	MP-G	124.0	1.016	0.0	0.0	1.43613
	RAN	124	1.016			0.67447
	LOC	124	1.016			0.72533
B/b07.stp - 111	PCC	111	1			0.00889
	CM	127	1.144			0.00168
	BP-E	111.0	1	0.0	0.0	0.20408
	BP-G	111.0	1	0.0	0.0	4.94428
	MP-E	111.0	1	0.0	0.0	2.28961
	MP-G	111.0	1	0.0	0.0	2.33191
	RAN	111	1			1.05543
	LOC	111	1			0.80671
B/b08.stp - 104	PCC	104	1			0.01663
	CM	111	1.067			0.00139
	BP-E	104.0	1	0.0	0.0	4.2071
	BP-G	104.0	1	0.0	0.0	0.32796
	MP-E	104.0	1	0.0	0.0	2.08557
	MP-G	104.0	1	0.0	0.0	1.9711
	RAN	104	1			0.75536
	LOC	104	1			0.65521
B/b09.stp - 220	PCC	221	1.005			0.07283
	CM	226	1.027			0.00125
	BP-E	220.0	1	0.0	0.0	0.47687
	BP-G	220.0	1	0.0	0.0	0.34666
	MP-E	220.5	1.002	0.25	0.5	3.69861
	MP-G	220.3	1.001	0.21	0.45826	4.44612
	RAN	220	1			1.8433
	LOC	220	1			1.34599
B/b10.stp - 86	PCC	98	1.139			0.01338
	CM	104	1.209			0.00154
	BP-E	93.2	1.084	2.76	1.66132	6.96276
	BP-G	92.0	1.070	7.4	2.72029	40.21622
	MP-E	90.1	1.048	3.29	1.81384	6.39866
	MP-G	90.2	1.049	3.16	1.77764	5.86142
	RAN	86	1			1.78942
	LOC	86	1			4.18135

Instance - OPT	Algorithme	Résultat	Ratio	Variance	Écart-type	Temps d'exécution
B/b11.stp - 88	PCC	93	1.057			0.02507
	CM	123	1.397			0.00183
	BP-E	91.6	1.40	0.64	0.8	10.33494
	BP-G	91.4	1.039	0.84	0.91652	24.65561
	MP-E	90.7	1.031	0.81	0.9	4.79364
	MP-G	90.5	1.028	0.65	0.80623	5.32104
	RAN	90	1.023			4.21122
	LOC	90	1.023			1.78671
B/b12.stp - 174	PCC	174	1			0.09496
	CM	181	1.040			0.0014
	BP-E	174.0	1	0.0	0.0	0.7846
	BP-G	174.0	1	0.0	0.0	4.98017
	MP-E	174.0	1	0.0	0.0	6.91646
	MP-G	174.0	1	0.0	0.0	8.1265
	RAN	174	1			5.16738
	LOC	174	1			4.10185
B/b13.stp - 165	PCC	175	1.061			0.02811
	CM	189	1.145			0.00266
	BP-E	175.0	1.061	0.0	0.0	4.02466
	BP-G	174.4	1.057	0.84	0.91652	16.34604
	MP-E	173.5	1.051	1.85	1.36015	5.27839
	MP-G	173.3	1.050	2.01	1.41774	7.1027
	RAN	175	1.061			4.46725
	LOC	171	1.036			5.42496
B/b14.stp - 235	PCC	238	1.013			0.04599
	CM	250	1.064			0.00203
	BP-E	236.9	1.008	0.09	0.3	4.92103
	BP-G	237.0	1.009	0.0	0.0	0.64313
	MP-E	236.5	1.006	0.45	0.67082	6.04964
	MP-G	236.4	1.006	0.24	0.4899	7.23236
	RAN	236	1.004			7.36811
	LOC	235	1			4.66581
B/b15.stp - 318	PCC	325	1.022			0.19289
	CM	333	1.047			0.00174
	BP-E	319.0	1.003	0.0	0.0	6.53916
	BP-G	319.0	1.003	0.0	0.0	4.85837
	MP-E	318.6	1.002	0.24	0.4899	8.16159
	MP-G	318.4	1.001	0.24	0.4899	9.37917
	RAN	319	1.003			8.64159
	LOC	318	1			4.02937
B/b16.stp - 127	PCC	137	1.079			0.06532
	CM	169	1.331			0.00629
	BP-E	135.5	1.213	0.65	0.80623	154.63724
	BP-G	135.8	1.069	0.36	0.6	71.22357
	MP-E	135.8	1.069	0.16	0.4	10.71581
	MP-G	135.8	1.069	0.16	0.4	9.68463
	RAN	133	1.047			6.75747
	LOC	134	1.055			5.14052

Instance - OPT	Algorithme	Résultat	Ratio	Variance	Écart-type	Temps d'exécution
B/b17.stp - 131	PCC	135	1.031			0.0631
	CM	146	1.114			0.00219
	BP-E	134.0	1.023	0.0	0.0	0.97911
	BP-G	133.4	1.018	0.84	0.91652	232.05891
	MP-E	133.0	1.015	0.8	0.89443	11.69888
	MP-G	133.0	1.015	1.0	1.0	10.24516
	RAN	133	1.015			12.37886
	LOC	131	1			8.81172
B/b18.stp - 218	PCC	222	1.018			0.24027
	CM	227	1.041			0.00179
	BP-E	219.8	1.008	0.36	0.6	16.5602
	BP-G	220.0	1.009	0.0	0.0	1.44029
	MP-E	218.3	1.001	0.41	0.64031	18.61842
	MP-G	218.9	1.004	0.69	0.83066	15.53312
	RAN	221	1.014			12.2912
	LOC	218	1			10.0428

Nous avons constaté que la recherche locale donne toujours le meilleur rapport avec OPT mais le temps d'exécution est plus lent que celui de l'algorithme génétique avec la population initiale RAN. Nous avons ensuite testé nos algorithmes avec le testset C ayant des instances plus grandes pour voir quelle est la meilleure méthode.

Instance - OPT	Algorithme	Résultat	Ratio	Temps d'exécution
C/c01.stp - 85	PCC	88	1.035	0.00981
	CM	134	1.576	0.0178
	RAN	88	1.035	5.20974
	LOC	88	1.035	28.57226
C/c02.stp - 144	PCC	144	1	0.03966
	CM	194	1.347	0.01812
	RAN	144	1	2.6162
	LOC	144	1	39.4302
C/c03.stp - 754	PCC	779	1.033	2.35076
	CM	887	1.176	0.02155
	RAN	774	1.028	83.03549
	LOC	757	1.004	905.82514
C/c04.stp - 1079	PCC	1105	1.024	5.78984
	CM	1163	1.078	0.01683
	RAN	1101	1.026	131.77565
	LOC	1089	1.009	902.8773
C/c05.stp - 1579	PCC	1604	1.016	22.05328
	CM	1662	1.053	0.01204
	RAN	1587	1.005	384.18035
	LOC	1579	1	1012.31697
C/c06.stp - 55	PCC	60	1.091	0.01355
	CM	142	2.582	0.02329
	RAN	56	1.018	8.27837
	LOC	55	1	32.65182

Instance - OPT	Algorithme	Résultat	Ratio	Temps d'exécution
C/c07.stp - 102	PCC	115	1.127	0.04561
	CM	186	1.824	0.02133
	RAN	103	1.010	19.24747
	LOC	115	1.127	69.87937
C/c08.stp - 509	PCC	534	1.049	3.55128
	CM	659	10295	0.0196
	RAN	521	1.024	124.92103
	LOC	515	1.012	2845.07758
C/c09.stp - 707	PCC	727	1.028	7.88967
	CM	859	1.215	0.01777
	RAN	726	1.027	285.42667
	LOC	712	1.007	2744.46915
C/c10.stp - 1093	PCC	1123	1.027	30.95651
	CM	1185	1.084	0.01521
	RAN	1106	1.012	532.90324
	LOC	1095	1.002	4611.81076
C/c11.stp - 32	PCC	35	1.094	0.03084
	CM	85	2.656	0.02522
	RAN	37	1.156	21.87121
	LOC	35	1.094	97.30895
C/c12.stp - 46	PCC	49	1.065	0.09542
	CM	79	1.717	0.02573
	RAN	48	1.043	30.71704
	LOC	47	1.022	304.05071
C/c13.stp - 258	PCC	275	1.066	6.24189
	CM	349	1.353	0.02818
	RAN	269	1.043	299.5505
	LOC	265	1.027	2495.9627
C/c14.stp - 323	PCC	338	1.046	14.18791
	CM	377	1.167	0.02518
	RAN	332	1.028	513.02548
	LOC	330	1.022	4206.61169
C/c15.stp - 556	PCC	574	1.032	57.08881
	CM	624	1.122	0.0197
	RAN	559	1.005	1193.97061
	LOC	557	1.002	3354.80823
C/c16.stp - 11	PCC	13	1.182	0.10379
	CM	53	4.81	0.0839
	RAN	12	1.091	21.66883
	LOC	13	1.182	154.92337
C/c17.stp - 18	PCC	19	1.056	0.39421
	CM	41	2.278	0.08398
	RAN	19	1.056	33.13428
	LOC	19	1.056	187.95379
C/c18.stp - 113	PCC	130	1.416	28.65958
	CM	189	1.673	0.05568
	RAN	121	1.071	521.30448
	LOC	118	1.044	6957.66043

Instance - OPT	Algorithme	Résultat	Ratio	Temps d'exécution
C/c19.stp - 146	PCC	160	1.096	57.41122
	CM	227	1.555	0.05771
	RAN	155	1.062	754.54337
	LOC	151	1.034	6625.0099
C/c20.stp - 267	PCC	269	1.007	239.8617
	CM	335	1.255	0.05104
	RAN	268	1.004	2062.32066
	LOC	268	1.004	1869.77677

Dans la plupart des cases, la recherche locale reste toujours très efficace avec un bon rapport d'approximation mais le temps d'exécution est très long (voire des heures!!!). Elle n'est plus applicable dans notre situation (timeout de 5 minutes).
Nous avons essayé une dernière fois avec 4 instances très grands de 1000 et 2500 nœuds.

Instance - OPT	Algorithme	Résultat	Ratio	Temps d'exécution
D/d01.stp - 106	PCC	107	1.009	0.0311
	CM	202	1.906	0.09079
	RAN	107	1.009	18.90095
	LOC	107	1.009	100.68533
D/d02.stp - 220	PCC	237	1.077	0.1186
	CM	332	1.510	0.06196
	RAN	232	1.510	24.21449
	LOC	237	1.077	150.35229
E/e01.stp - 111	PCC	125	1.126	0.16051
	CM	260	2.342	0.53434
	RAN	125	1.126	44.82023
	LOC	125	1.126	771.14907
E/e02.stp - 214	PCC	255	1.192	0.30169
	CM	454	2.121	0.35598
	RAN	241	1.126	61.92313
	LOC	255	1.192	1088.20558

5 Conclusion

Les opérateurs de sélection et d'initialisation de la population donnent une bonne base pour la résolution du problème. Si le critère est aussi de minimiser le temps d'exécution il faut choisir l'algorithme génétique perturbé sur les poids. L'introduction de nouveaux opérateurs de croisement serait une bonne idée pour optimiser la solution du problème.