UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE

FACULTE D'ÉLECTRONIQUE ET D'INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

Mini-Projet de e-commerce

Rapport de projet

Résolution de problème de définition du gagnant dans des enchères combinatoires en utilisant la métaheuristique Gray Wolf Optimisation

Réalisé par :

- ABDELALI Asma Nihad (Groupe: 1)

Responsable du module : D. Boughaci

Année Universitaire 2020-2021

I. Introduction :

Le problème de la détermination du gagnant dans les enchères combinatoires, ou bien **Winner determination problem** (WDP) consiste à trouver un ensemble d'objets qui satisfaisant plusieurs enchères à objets multiples sans conflit. C'est un problème NP-Complet qui ne peut être résout par des algorithmes classiques qu'en un long temps et de grandes ressources mémoire, pour cela il est préférable d'utiliser des métaheuristiques qui sont plus performantes dans ce domaine.

Dans ce mini-projet le WDP en utilisant la métaheuristique Gray Wolf Optimizer ou GWO développée par Seyedali Mirjalili.

II. Définition du problème de détermination du gagnant dans les enchères combinatoires

Les enchères combinatoires consistent en un ensemble d'offres sur multiples objets à la fois, chaque offre dispose d'un prix ou valeur, et le but c'est de déterminer les offres à satisfaire, en faisant attention aux conflits ainsi qu'en maximisant notre gain, soit, la somme des valeurs des offres.

En résumé :

Nous disposons de N items et de M offres (auctions / Bids).

Chaque offre est composée de i Items ayant la valeur V.

Le but est de maximiser la somme des V sous la contrainte que chaque item ce trouve dans une seule offre au plus.

III. Implémentation de l'approche Espace des états pour le problème WDP :

- L'espace de recherche c'est l'ensemble de toutes les enchères
- Une solution est un ensemble d'enchères et aussi la proie recherchée par les loups.
- La meute de loups représente un ensemble de solutions, les trois meilleures solutions sont respectivement l'alpha, le beta et delta, le reste des loups seront des omégas.
- L'ensemble des enchères (WDP) est représenté par une liste de Bids (enchères).
- Chaque enchère détient une liste de type Short qui représente les Items (objets).
- La fonction de fitness est la maximisation de la somme des gains des enchères du problème.
- Dans le monde réel les loups se repèrent à leurs sens cependant il n'est pas possible de voir la solution (proie) dans un monde mathématique, nous assumons alors que l'alpha, le bêta et le delta ont la meilleure connaissance de la position de la proie.

IV. Mise à jour de la position.

$$D = |C.X_p(t) - X(t)|$$
$$X(t+1) = X_p(t) - A.D$$

A et C sont des vecteurs de coefficients, $X_p(t)$ est la position de la proie à l'instant t et X(t) indique la position du loup actuel à l'instant t.

$$A = 2.a.r1 - a$$

Cette équation fera que -a<A<a.

A oblige le loup de se rapprocher ou de s'éloigner de la solution. '**a**' est initialement égal à 2 et décrémenté au long des itérations, r1 et r2 sont des nombres au hasard, ils permettent au loup de se repositionner.

$$C = 2.r2$$

Donc C aura les valeurs entre 0 et 2.

C>1 plus d'exploration, C<1 plus d'exploitation.

$$X_1 = X_\alpha - A_1.D_\alpha$$
$$X_2 = X_\beta - A_2.D_\beta$$
$$X_3 = X_\delta - A_3.D_\delta$$

Pour trouver la position à l'instant prochain il suffit de calculer

$$X(t+1) = \frac{X_1 + X_2 + X_3}{3}$$

Ces équations nous permettent de se positionner dans un espace à trois dimensions, dans le monde réel ces repositionnements hasardeux sont dus au terrain et aux obstacles.

Par contre dans un WDP on ne peut pas faire ces multiplications sur un ensemble d'enchères, nous utilisons la position afin de modifier une partie de la solution trouvée.

La modification se fait par la suppression d'une partie des enchères de la solution et leur remplacement pas d'autres au hasard qui respectent les conflits entre objets

Il se peux qu'elle soit meilleure ou moins bien que la précédente.

Si la solution d'un individu ne s'est pas améliorée après une itération, une nouvelle solution est générée.

V. Solution au hasard

Comme on ne peut pas générer une solution random de la façon habituelle tout en respectant les conflits entre les objets **The Random Key Encoding** est habituellement utilisé, dans ce mini-projet une variante a été utilisée.

Le **Random Key Encoding** consiste en un ensemble de réels qui représentent l'ordre dans lequel les offres vont êtres considérées.

Par exemple pour un ensemble d'offres B_1, B_2, B_3 et un r = {0.6, 0.85, 0.23}.On va considérer les Bids selon l'ordre suivant : B_2, B_1, B_3.

B_2 va être rajouté automatiquement à la solution ensuite B_1 si et seulement si il ne cause pas de conflit pas avec B_2 et ainsi de suite.

VI. Pseudo Code

Gray Wolf Optimization Pseudo-Code

Entrée: Taille de la meute, maxIterations : Entier;

Sortie: Ensemble d'enchères.

Debut

```
wolfPack.Initialiser(Taille de la meute); // En utilisant le Random Key Encoding
Alpha = Collections.Max(wolfPack); wolfPack.remove(Alpha);
Beta = Collections.Max(wolfPack); wolfPack.remove(Beta);
Delta= Collections.Max(wolfPack); wolfPack.remove(Delta);
Initialize a, r1, r2, A, C; //En utilisant les équations vues précédemment
While(iter<maxIteration){
        for (wolf: wolfPack) {
                wolf.calculerPosition();
                wolf.majSolution();
        }
        a = 2 - iter* ((2.0) / maxIterations);
        Update r1, r2, A, C;
        Update Alpha, Beta, Delta; // Remettre Alpha, Bêta, Delta dans la meute et
        récupérer les nouveau 3 meilleures solutions
        iter ++;
}
return Alpha;
```

Fin;

```
Procédure UpdateSolution()
Var : i, cpt, encheresAremplacer : Entier
Début
    Wolf w = this;
    encheresAremplacer = abs(position) % w.nombreEncheres; // Le nombre d'enchères dans cette solution.
        for (int i=d; d<w.nombreEncheres; d++){// vider une partie des enchères et actualiser les Conflicts
        w.Encheres.set(i, null);
        w.actualiserListeConflicts();
    }
    for(i =0, cpt=0; j<nombreEncheresTotal cpt<encheresAremplacer ; j++){
        if( ! EncheresTotales.get(i).estEnConflict(this.Encheres){
        this.AjouterEnchère( EncheresTotales.get(i)) // En admettant la fonction Ajouter met à jour le total des gains et la liste des conflits
        cpt ++;
        }
    }
    if (this.getGain() < w.getGain())// Si la nouvelle solution est meilleure alors la prendre
        this=w;
    else //Sinon générer une nouvelle solution
        this.générerRandom()
Fin.
```

VII. Expérimentations :

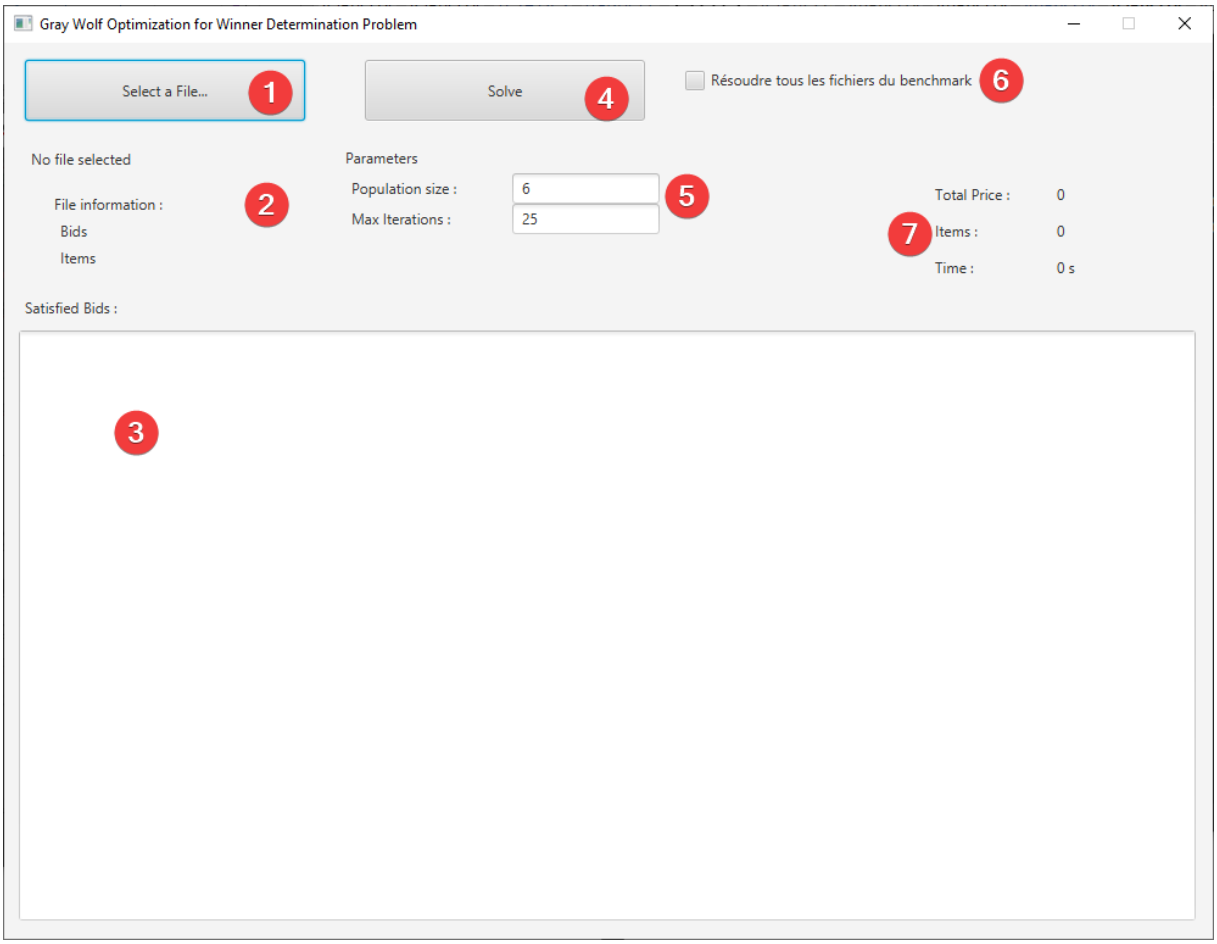
1) Environnement expérimental :

Processeur : Intel® Core™ i5-7300HQ CPU @ 2.50GHz

Ram : 8.00 Go (DDR4, 2400 MHz)

Système d'exploitation : Windows 10 Famille 64 bits, processeur x64

2) Interface Graphique :



1. Bouton pour sélectionner une instance à résoudre
2. Informations sur le fichier sélectionné
3. Solution trouvée : un ensemble d'enchères et les objets que chacune contient
4. Bouton pour résoudre
5. Paramètres à préciser pour l'algorithme
6. Pour résoudre le reste des instances se trouvant dans le même fichier que l'instance sélectionnée. Le résultat sera sauvegardé dans un fichier 'resultat.csv'.

3) Résultats des tests sur des Benchmarks

Les tests des algorithmes ont été effectués sur le benchmark RL 1500-1500 groupe 5 :

#instances	# Nombre d'items mis à la vente	#enchères
100	1500	1500

i. Résultats sur les benchmarks de 601 à 700 :

Le test complet du groupe du benchmark avec ces paramètres a pris environ 30 minutes.

Benchmark	Itérations	Individus	Gain GWO	Temps GWO
601	25	12	89466,124	12,636
601	15	20	93321,726	17,302
602	15	20	90641,245	15,562
603	15	20	85111,939	15,082
604	15	20	91173,721	15,283
605	15	20	97355,92	16,212
606	15	20	86919,816	15,408
607	15	20	93457,209	15,387
608	15	20	86817,751	16,29
609	15	20	92165,231	17,049
610	15	20	93940,254	16,943
611	15	20	88496,63	12,513
612	15	20	90199,532	17,868
613	15	20	87132,482	13,796
614	15	20	90398,244	13,28
615	15	20	92051,815	14,531
616	15	20	87954,572	12,811
617	15	20	89872,156	14,986
618	15	20	87497,614	14,167
619	15	20	84796,725	13,33
620	15	20	94906,142	13,732
621	15	20	91378,174	10,833
622	15	20	86987,245	14,658
623	15	20	93660,037	14,154
624	15	20	90143,856	13,788
625	15	20	82920,059	14,3
626	15	20	95859,123	11,655
627	15	20	95143,965	10,806
628	15	20	91807,155	15,1
629	15	20	92571,85	13,936
630	15	20	86860,932	15,54
631	15	20	97811,561	14,832
632	15	20	84530,238	14,298
633	15	20	89952,631	13,762
634	15	20	90766,37	11,325
635	15	20	87514,139	13,614
636	15	20	89472,405	14,293
637	15	20	90331,638	14,092
638	15	20	91154,214	13,112
639	15	20	90678,264	12,786
640	15	20	88197,63	13,998
641	15	20	97587,862	13,717
642	15	20	91247,25	13,803
643	15	20	89503,069	14,305

644	15	20	94474,923	14,107
645	15	20	95481,955	13,984
646	15	20	93825,043	13,449
647	15	20	88617,624	15,07
648	15	20	88669,962	16,936
649	15	20	92654,765	14,993
650	15	20	89602,837	12,723
651	15	20	83546,373	13,869
652	15	20	88563,424	12,976
653	15	20	93508,452	16,23
654	15	20	90749,218	15,091
655	15	20	94537,073	13,941
656	15	20	89245,357	14,294
657	15	20	87787,637	12,686
658	15	20	97875,855	12,25
659	15	20	88072,049	12,619
660	15	20	91687,376	15,036
661	15	20	92531,559	15,638
662	15	20	86724,318	14,991
663	15	20	88082,751	13,355
664	15	20	94220,517	14,824
665	15	20	92922,574	13,059
666	15	20	87772,998	13,824
667	15	20	91509,739	12,428
668	15	20	90015,231	14,602
669	15	20	90165,595	13,488
670	15	20	91220,172	12,043
671	15	20	87783,648	15,054
672	15	20	85574,679	12,74
673	15	20	95794,462	14,325
674	15	20	86657,617	15,199
675	15	20	92765,347	11,826
676	15	20	91265,3	12,5
677	15	20	90488,551	14,14
678	15	20	82140,058	13,566
679	15	20	89134,919	11,531
680	15	20	91256,357	11,225
681	15	20	87119,844	14,368
682	15	20	89824,684	12,566
683	15	20	88848,998	12,142
684	15	20	86341,32	14,957
685	15	20	87499,234	14,57
686	15	20	90626,876	14,239
687	15	20	89923,993	14,409
688	15	20	94320,59	14,737

689	15	20	86242,03	14,38
690	15	20	91223,225	11,689
691	15	20	92198,309	12,968
692	15	20	97633,851	14,271
693	15	20	90922,469	13,702
694	15	20	90797,853	15,771
695	15	20	92135,418	16,099
696	15	20	94623,576	14,035
697	15	20	93483,775	11,766
698	15	20	90887,033	14,917
699	15	20	93209,855	15,757
700	15	20	86421,447	12,476
701	15	20	86421,447	12,476

ii. Comparaison avec d'autres algorithmes testés sur les mêmes instances.

GWO a été exécuté avec les paramètres :

- 15 individus et 20 itérations.
- 12 individus et 50 itération et finalement.
- 25 individus et 50 itérations.

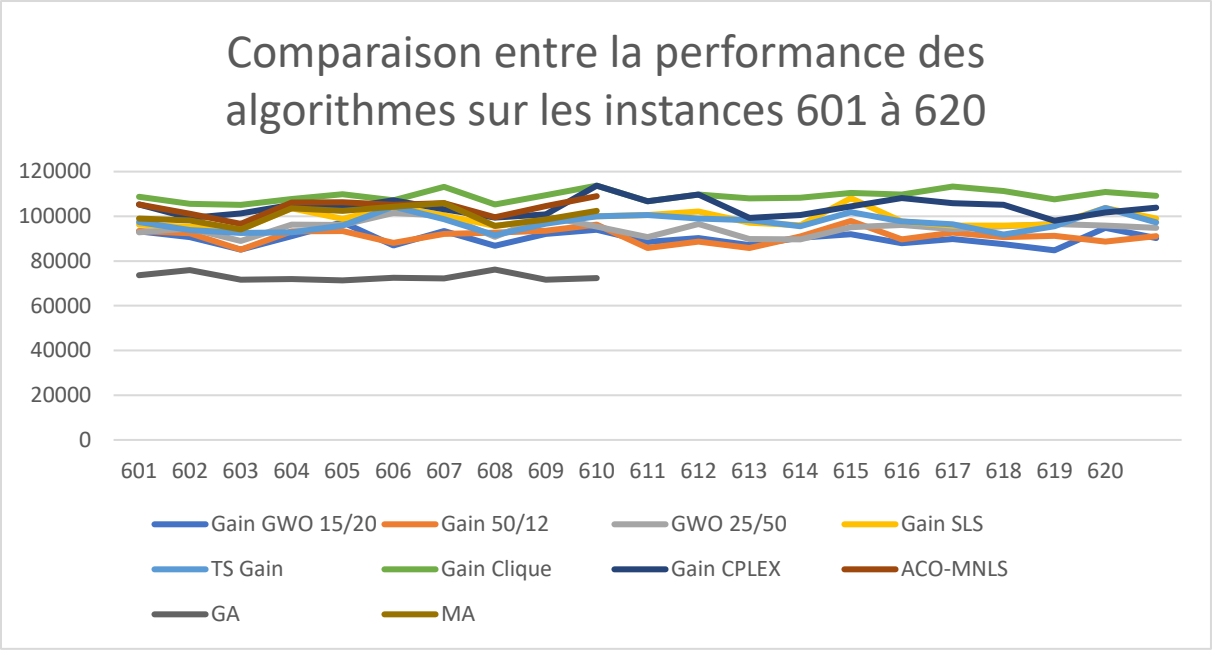
Les algorithmes testés sont :

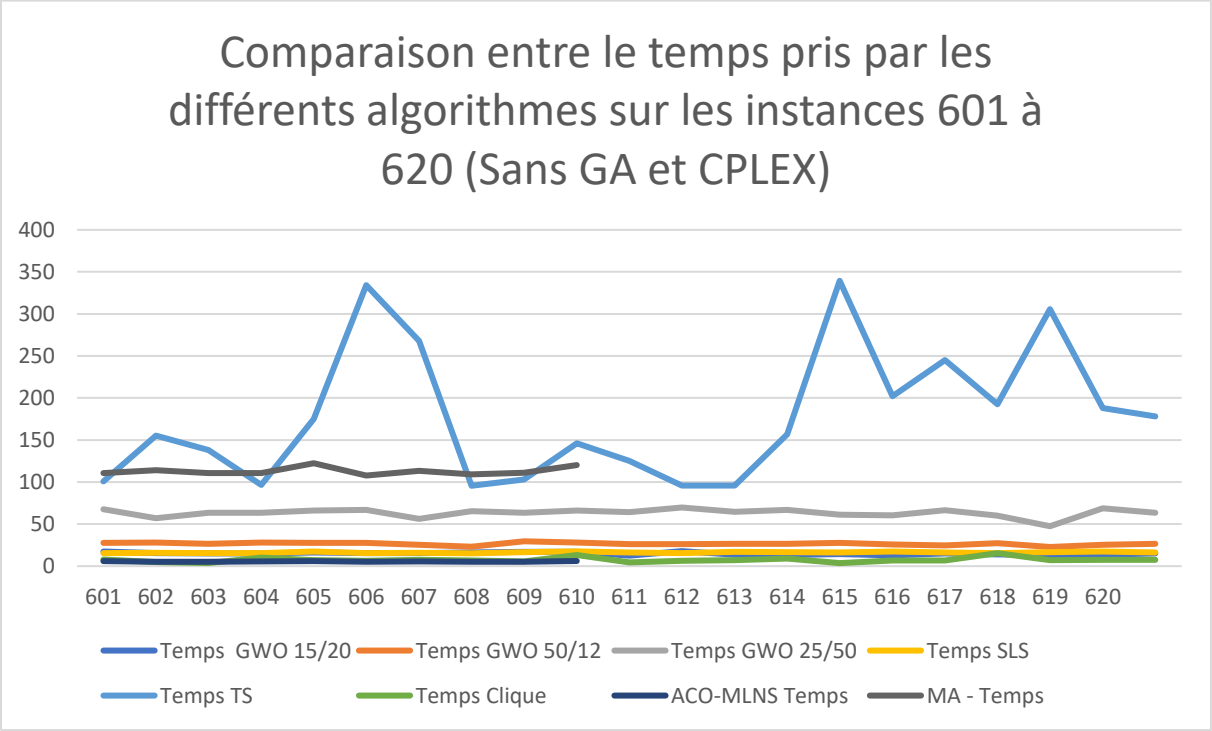
- Recherche locale stochastique (SLS) [1]
- La recherche tabou (TS) [1]
- MaxWClique algorithm [2]
- CPLEX 12.4 solver [2]
- Algorithme génétique (GA) [3]
- Memetic Algorithme (MA) [3]
- A hybrid Ant Colony Optimization with a novel Multi-Neighborhood Local Search (ACO-MNLS) [3]

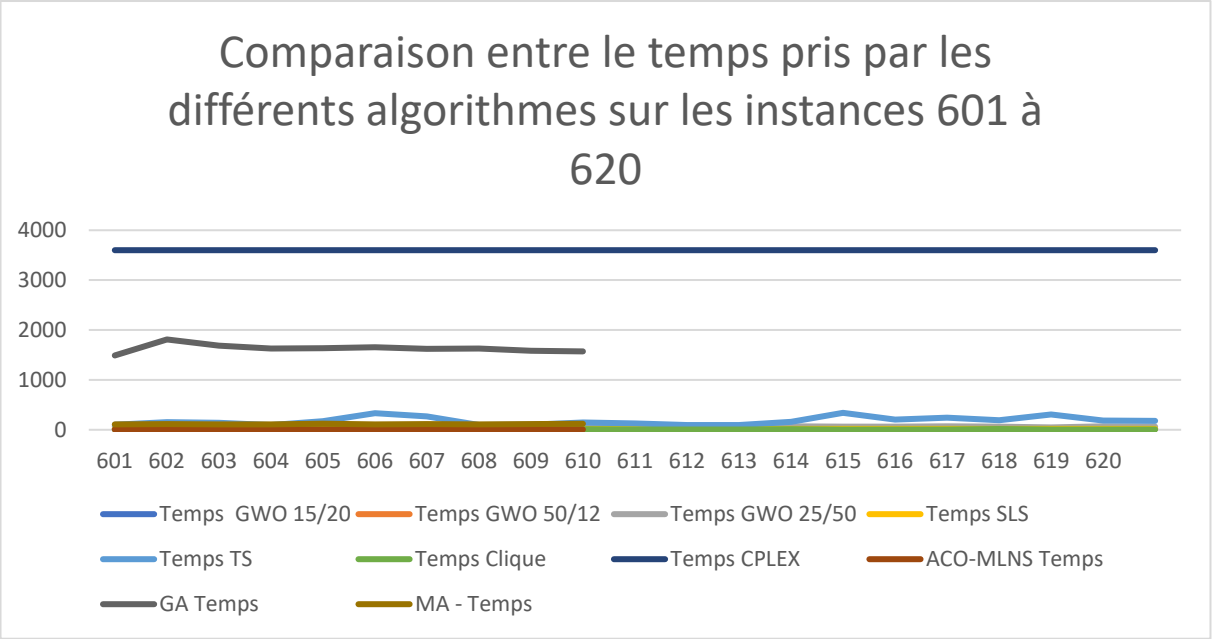
Les résultats sont dans le fichier Excel joint à celui-ci par manque d'espace.

Voici les graphes comparatifs mais ils ont meilleure visibilité sur le fichier Excel.

L'Algorithme génétique et GA et le solveur CPLEX prennent beaucoup de temps alors ils ont été exclus de l'un des graphes pour une meilleure comparaison.


Comparaison entre la performance des algorithmes sur les instances 601 à 620
120000
100000
80000
60000
40000
20000
0
601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620
Gain GWO 15/20
Gain 50/12
GWO 25/50
Gain SLS
TS Gain
Gain Clique
Gain CPLEX
ACO-MNLS
GA
MA


Comparaison entre le temps pris par les différents algorithmes sur les instances 601 à 620 (Sans GA et CPLEX)
400
350
300
250
200
150
100
50
0
601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620
Temps GWO 15/20
Temps GWO 50/12
Temps GWO 25/50
Temps SLS
Temps TS
Temps Clique
ACO-MLNS Temps
MA - Temps



4) Interprétation des résultats :

Nous remarquons que GWO, selon ses paramètres performe différemment, on peut dire que les résultats obtenus approchent celles de la recherche taboue. La performance peut probablement être améliorée en sacrifiant du temps.

Coté temps : Si on exclut les algorithmes très longs comme le solveur CPLEX et GA, GWO avec des paramètres équilibrés il prend le même temps que SLS, il plus rapide que l'algorithme de la clique et ACO, avec plus d'individus et d'itérations il ne prend pas beaucoup plus de temps.

Coté performance : on peut dire qu'il est dans la moyenne, il n'est pas assez performant que l'algorithme de la clique mais ses variations de temps sont dans la même intervalle que TS et SLS.

Mais ces tests n'ont pas été fait par manque de temps. Avec plus d'itérations et d'individus l'algorithme prendra autant de temps que le CPLEX qui a été limité à 1h et donnera de meilleurs résultats.

5) Développement futur

i. Graphe de conflit

Un graphe de conflit est un graphe qui représente les conflits entre les enchères d'une instance, dû à un manque de temps, une représentation graphique n'a pas pu être implémentée. Un graphe de conflits est un graphe qui a comme nœuds représentés par

des enchères et chaque arc entre deux enchères représente au moins un item en commun entre les deux enchères.

ii. Utiliser le parallélisme

Utiliser un système multi agent ou chaque individu est un agent ou les threads afin de mieux exploiter les ressources de la machine.

6) Conclusion Générale :

En sachant que les enchères représentent une partie importante du commerce qu'il soit électronique ou non, il est important d'optimiser le temps de résolution du problème des enchères combinatoires.

Un problème Np-complet tel que les enchères combinatoires pourraient prendre beaucoup de temps à des algorithmes classiques comme la recherche Taboue ou même les algorithmes exacts développées spécifiquement pour résoudre ce problème.

7) Références :

[1] Dalila Boughaci, Belaïd Benhamou, Habiba Drias : Une recherche locale stochastique pour le problème de la détermination du gagnant dans les enchères combinatoires.

[2] Qinghua Wu a & Jin-Kao Hao: A clique-based exact method for optimal winner determination in combinatorial auctions.

[3] M. B. Dowlatshahi and V. Derhami: Winner Determination in Combinatorial Auctions using Hybrid Ant Colony Optimization and Multi-Neighborhood Local Search.