Une recherche locale stochastique pour le problème de la détermination du gagnant dans les enchères combinatoires

Dalila Boughaci, Belaïd Benhamou, Habiba Drias

Une recherche locale stochastique pour le problème de la détermination du gagnant dans les enchères combinatoires

D.Boughaci[1,2], B.Benhamou[1], H.Drias[2]

[1]INCA/LSIS, CMI 39 rue Fredric Joliot-Curie Marseille 130013.
[2]LRIA/USTHB- BP 32 El-Alia, Beb-Ezzoaur, 16111, Alger, Algérie.
boughaci@cmi.univ-mrs.fr, dboughaci@usthb.dz

Résumé

Dans ce papier, nous proposons un algorithme de recherche locale stochastique (SLS) pour résoudre le problème de la détermination du gagnant (PDG) dans les enchères combinatoires. Des expérimentations numériques sont réalisées sur des benchmarks de diverses tailles dans le but de tester et de prouver l'efficacité de notre approche. Les résultats trouvés par la méthode SLS sont nettement meilleurs que ceux fournis par les algorithmes de recherche tabou et Casanova.

Abstract

In this paper, a stochastic local search (SLS) is studied for the optimal winner determination problem (WDP) in combinatorial auctions. Experiments on realistic data sets of various sizes are performed to show and compare the effectiveness of our approach. The computational experiments show that the SLS method provides competitive results and finds solutions of a higher quality than Tabu search and Casanova methods.

1 Introduction

Un mécanisme classique de vente aux enchères consiste en un vendeur souhaitant maximiser son gain et un ensemble d'acheteurs souhaitant minimiser leur perte en fonction de l'estime qu'ils portent à l'objet de la vente.

Une enchère est un mécanisme qui permet d'allouer les ressources (objets, services) aux acteurs du commerce électronique (vendeurs/acheteurs) en se basant sur des règles prédéfinies. Ces règles définissent le processus d'échange de propositions, la détermination du gagnant et l'accord final.

Parmi les enchères les plus largement dominantes, nous citons : les enchères anglaises, les enchères hollandaises, les enchères à enveloppe scellées, et les enchères de Vickrey [12].

Face à ces enchères qui portent uniquement sur le prix et qui manipulent qu'un seul objet, d'autres mécanismes ont été proposés tels que les enchères combinatoires.

Dans ce papier, on s'intéresse aux enchères combinatoires : un ensemble d'objets est soumis à la vente face à plusieurs acheteurs, chaque acheteur, pour des raisons de complémentarité entre les objets, désire acheter un sous-ensemble d'objets qui lui est propre, et dont il fournit une estimation. Les conflits entre les acheteurs naissent des éventuelles intersections entre les sous-ensembles. Le vendeur doit alors résoudre un problème d'optimisation combinatoire NP-complet pour réaliser la vente qui lui rapportera le plus.

Les enchères combinatoires ont plusieurs applications notamment en économie, la théorie des jeux et l'allocation des ressources dans les systèmes multi-agents [18, 16, 17]. Les enchères combinatoires permettent une allocation meilleure des objets selon les besoins spécifiques des enchérisseurs. La difficulté de modéliser et de formaliser ce type d'enchère apparaît dans le nombre de combinaisons d'objets possibles. Pour cela, ce type d'enchères n'a été utilisé que rarement, dans des applications bien spécifiques [13]. Les enchères de la FCC [1] sont un exemple réel de cette catégorie d'enchères combinatoires.

[1]http ://wireless.fcc.gov/auctions.

Dans ce papier, nous proposons un algorithme de recherche locale stochastique pour résoudre le problème de la détermination du gagnant (PDG). Des expérimentations numériques sont réalisées sur des benchmarks de diverses tailles dans le but de tester et de prouver l'efficacité de notre approche.

Le reste de ce papier est organisé comme suit : la prochaine section décrit le mécanime d'enchères combinatoires et donne un aperçu sur quelques méthodes de résolution. La troisième section propose un algorithme de recherche locale au problème de la détermination du gagnant dans les enchères combinatoires. Quelques résultats expérimentaux sont donnés dans la quatrième section. Enfin, la cinquième section donne une conclusion et quelques perspectives de ce travail.

2 Enchères combinatoires

Une enchère est un mécanisme important pour l'allocation des ressources dans les systèmes multi-agents. Une enchère combinatoire est une enchère multi-objets permettant la soumission d'offres sur une combinaison d'objets. Les prix proposés par un enchérisseur pour un objet dépendent alors des autres objets. On distingue deux cas possibles [14, 11] :

1. le cas où les objets sont complémentaires. Par exemple un acheteur peut évaluer une unité centrale au prix x et un écran au prix y, mais il évalue la paire unité centrale et écran au prix $z > x + y$. Ici, les objets écran et unité centrale sont complémentaires.
2. Le cas où les objets sont substituts l'un de l'autre : le cas où un acheteur peut évaluer une seule copie d'un livre à un prix x mais deux copies du même livre à un prix $z < 2.x$. Dans ce cas, les deux objets sont substituts l'un de l'autre.

2.1 Modélisation du problème

Le PDG est un problème complexe et équivalent au problème d'emballage d'ensemble qui est NP-Complet [14, 6].

Le problème de la détermination du gagnant dans les enchères combinatoires peut être énoncé comme suit :

Considérons un ensemble de m objets, M= $\{1, 2 \ldots m\}$ à vendre aux enchères et un ensemble de n offres d'achat N= $\{1, 2 \ldots n\}$. Chaque offre j est un tuple $<S_j, P_j>$, où S_j ($S_j \sqsubseteq M$) est l'ensemble d'objets couverts par l'offre j, P_j étant son prix ($P_j \succeq 0$).

Soit $a_{(m \times n)}$ une matrice binaire de m lignes et n colonnes tel que a_{ij}=1 si l'objet $i \in S_j$, 0 autrement.

De plus, soient les variables de décision x_j, x_j=1 si l'offre j est acceptée (une offre gagnante), 0 autrement (une offre perdante).

Le problème de la détermination du gagnant consiste à trouver les offres gagnantes qui maximisent le gain du vendeur sous la contrainte que chaque objet ne peut être affecté qu'à un seul acheteur. Le PDG peut être formalisé comme suit [16, 14] :

$$\textbf{Maximiser} \quad \sum_{j \in N} P_j.x_j \tag{1}$$

$$\textbf{Sous les contraintes} \ : \ \sum_{j \in N} a_{ij}.x_j \leq 1, \ \ i \in M \tag{2}$$

$$N = \{1, 2, \ldots, n\}, \ M = \{1, 2, \ldots, m\}, \ x_j \in \{0, 1\} \tag{3}$$

Où la fonction objectif (1) permet de maximiser le gain du vendeur calculé par la somme des prix des offres gagnantes et les contraintes interdisent qu'un objet figure dans deux offres différentes.

Exemple 1 *Exemple illustratif :*

Cet exemple permettra de mieux comprendre les notations utilisées dans la modélisation du problème PDG. Considérons un ensemble de cinq objets $\{1, 2, 3, 4, 5\}$ à vendre aux enchères et quatre offres d'achat. Chaque offre est représentée par un couple (S_j, P_j) où P_j indique le prix que l'enchérisseur propose à payer pour son offre j contenant l'ensemble d' objets S_j.

Soient les offres d'achat suivantes :

- *Offre 1 :* $\{(1, 2), 350\}$
- *Offre 2 :* $\{(1, 2, 3), 300\}$
- *Offre 3 :* $\{(3, 4, 5), 400\}$
- *Offre 4 :* $\{(4, 5), 200\}$

L'offre 1 par exemple contient un ensemble de deux objets (1, 2) qui vaut 350. L'offre 3 vaut 400 pour un ensemble de trois objets (3,4,5).

Les offres 1 et 3 peuvent constituer une allocation gagnante maximisant le gain du vendeur et dont le prix total de la vente vaut 750.

2.2 Travaux antérieurs

Plusieurs méthodes ont été proposées pour résoudre le problème de la détermination du gagnant [17]. Ces méthodes peuvent être divisées en deux catégories : les méthodes exactes et les méthodes approchées.

Les méthodes exactes sont basées généralement sur l'algorithme de Branch-and-Bound alors que les méthodes approchées sont basées sur les heuristiques ou

les méta-heuristiques. En effet, les métaheuristiques sont des méthodes approchées très utiles pour trouver une solution optimale pour les problèmes de grande taille et dont la méthode exacte n'arrive pas à trouver la solution exacte en un temps raisonnable.

Plusieurs algorithmes ont été développés pour trouver la solution optimale pour le PDG. Parmi eux, nous citons : Branch-on-Items (BoI) [16], Branch-on-Bids (BoB) [16], et (CABoB) [15]. CASS (Combinatorial Auction Structural Search) est un algorithme Branch-and-Bound pour le PDG proposé dans [4]. Dans [11], les auteurs ont proposé CAMUS (Combinatorial Auctions Multi-Unit Search). Les auteurs du papier [14] ont proposé une approche de programmation dynamique. Nisan a proposé une méthode de programmation linéaire [13]. Anderson *et al.* ont développé un algorithme basé sur la programmation en nombre entier [1]. Holland et O'sullivan ont employé la programmation par contrainte pour résoudre une enchère combinatoire particulière de Vickrey [7].

D'autre part, quelques méthodes approchées ont été étudiées pour le PDG. Nous citons : le recuit simulé hybridé avec un Branch-and- Bound (SAGII) [5] [6], la recherche locale Casanova [8] et plus récemment l'adaptation de la méthode mémetique au problème PDG [3].

3 Présentation de notre approche

La recherche locale démarre d'une solution initiale possible et essaie de l'améliorer, en cherchant une solution meilleure dans le voisinage courant. Un voisinage d'une solution A correspond à des éléments adjacents à A dont chacun est atteint par un changement dans la configuration courante. Le processus de recherche est réitéré jusqu'à ce qu'aucune amélioration dans la solution courante ne pourrait être faite.

La méthode de recherche locale que nous proposons utilise le codage de clef aléatoire, RK (Random Key Encoding) introduit par Bean pour les problèmes d'ordonnancement [2]. Ce codage nous permet de manipuler des solutions admissibles. Par conséquent, aucune pénalité additionnelle n'est nécessaire pour éliminer les solutions inadmissibles.

Plus précisement, notre recherche locale démarre d'une solution aléatoire et créée selon le codage RK, puis une solution voisine est sélectionnée pour être la prochaine solution courante. À chaque itération une offre non satisfaite [2] est sélectionnée pour inclusion dans l'allocation et toute offre contradictoire, pouvant se produire dans l'allocation courante quand de nouvelles offres sont considérées, doit être enlevée. La sélection de l'offre à inclure dans l'allocation courante se fait selon les deux critères suivants :

- Le premier critère consiste à choisir l'offre à considérer d'une manière aléatoire avec une probabilité fixe $wp > 0$.
- Le deuxième critère consiste à choisir, avec une probabilité $1 - wp$, la meilleure offre (celle maximisant le gain du vendeur une fois insérée) pour être considérée.

Notons que les offres sont en conflit si elles partagent un objet. Pour ne manipuler que des allocations admissibles tout au long du processus de recherche locale, nous définissons un graphe de conflit où les sommets sont les offres et les arcs relient les offres qui ne peuvent pas être acceptées ensemble. Ce graphe est utile pour enlever toutes les offres contradictoires[3] qui peuvent se produire dans les allocations courantes quand de nouvelles offres sont considérées.

Le processus de recherche est réitéré pour un certain nombre d'itérations fixé d'une manière empirique.

3.1 Représentation de la solution

Une solution est une collection d'offres satisfaisant les contraintes du problème et le cas échéant optimisant la fonction objectif. Nous utilisons un vecteur A de taille variable (ne dépassant pas le nombre d'offres) pour représenter une telle solution, où chaque composante A_i reçoit le numéro de l'offre gagnante.

La solution initiale de la recherche locale que nous proposons est générée aléatoirement suivant le codage RK qui fonctionne comme suit :

nous générons une suite r de n nombres réels aléatoires où n est le nombre d'offres soumises. Pour chaque offre j une valeur de clef r_j lui est associée et qui constitue son ordre de sélection. La première offre à choisir et à inclure dans l'allocation est celle ayant la plus grande valeur de clef. Ensuite, l'offre ayant la deuxième plus grande clef est acceptée dans la collection si son acceptation ne crée aucun conflit avec les offres déjà sélectionnées dans la collection courante, autrement elle est rejetée. Et ainsi de suite, jusqu'à ce que les n offres soient examinées. Nous obtenons un sous-ensemble d'offres qui peut être une solution au PDG.

Pour produire une solution admissible pour l'exemple illustratif 1, nous suivons les étapes suivantes :

[2] Une offre qui n'apparait pas dans l'allocation courante est dite non satisfaite. Elle devient satisfaite une fois qu'elle sera choisie et considérée dans l'allocation prochaine.

[3] Les offres contradictoires sont celles qui ont au moins un objet en commun.

– Produire d'abord une suite de quatre nombres réels aléatoires, par exemple, r= {0.65, 0.70, 0.80, 0.60}.
– La première offre à accepter est l'offre 3 ayant la plus grande clef (0.80). On aura donc l'allocation A= (3).
– Puis, l'offre ayant la deuxième plus grande clef est 2, mais elle est écartée parce qu'elle est en conflit avec l'offre 3 actuellement dans A. En effet, elles partagent l'objet 3.
– Ensuite, l'offre 1 peut être ajoutée à l'allocation A car elle ne présente aucun conflit avec les offres dans A. A=(3, 1).
– Puis, l'offre 4 est à écarter car les objets 4 et 5 sont déjà couverts par les offres de l'allocation courante A.

Nous obtenons alors l'allocation A= (3, 1) qui peut être une solution au problème du PDG considéré. Le prix global est tout simplement la somme des prix des offres gagnantes : 400 + 350 = 750.

A partir de l'allocation A, nous pouvons voir que les objets 1 et 2 sont affectés à l'enchérisseur 1 et les objets 3, 4 et 5 sont affectés à l'enchérisseur 3. Les offres gagnantes sont : 3 et 1. Les offres 2 et 4 sont des offres perdantes. Pour cet exemple, nous remarquons que tous les objets sont vendus.

3.2 Graphe de conflit

Pour ne manipuler que des solutions admissibles, nous avons créé un graphe de conflit où les sommets sont les offres et les arcs relient les offres qui ne peuvent pas être acceptées ensemble. Ce graphe permet de détecter les offres en conflit (i.e. les offres qui partagent au moins un objet). Grâce à ce graphe nous pouvons assurer et maintenir la faisabilité de nos allocations.

3.3 Critère d'Arrêt

Le processus de recherche est stoppé après un certain nombre d'itérations (max_iter) fixé d'une manière empirique.

3.4 L'algorithme de recherche locale pour le PDG

Une version de l'algorithme de la recherche locale pour le PDG est donnée dans l'algorithme 1.

Algorithm 1 : recherche locale stochastique pour le PDG.

1: **Entrée :** une instance de PDG, max_iter, wp
2: **Sortie :** une allocation A
3: Créer le graphe de conflit ;
4: $iter$ =0 ;
5: Générer aléatoirement une solution initiale A selon le RK ;
6: **Pour** $I = 1$ **à** max_iter
7: **Faire**
8: r $\Leftarrow$ un nombre aléatoire entre 0 et 1 ;
9: **Si** (r $< wp$) **alors** (*$Step\ 1$)
bid = une offre choisie aléatoirement
Sinon *bid=la meilleure offre ; (*Step 2)
10: **FinSi**
11: A= A avec l'offre *bid* sélectionnée ;
12: Enlever toute offre incompatible ;
13: **Fait**
14: **Retourner** la meilleure allocation trouvée.

4 Résultats expérimentaux

Dans le but de valider notre approche, des tests ont été effectués sur différents problèmes de diverses tailles. Nos algorithmes ont été implémentés en C sous Linux et exécutés sur la machine Pentium- IV 2.8 GHz, 1GO de RAM.

4.1 Benchmarks

Récemment Lau et Goh ont fourni de nouvelles données de diverses tailles ayant jusqu'à 1500 objets et 1500 offres [10]. Ces benchmarks tiennent compte de plusieurs facteurs tels : le facteur des prix, le facteur de préférence des enchérisseurs et les facteurs de distribution des objets sur les offres.

Dans nos expérimentations, nous utilisons les données pré-générées dans [10] pour lesquelles le CPLEX n'arrive pas à trouver la solution optimale en un temps raisonnable [6]. Cette batterie de test contenant 500 instances est disponible sur la page Web dont l'URL est : HTTP :/logistics.ust.hk/ zhuyi/instance.zip. Ces instances peuvent être divisées en 5 groupes de problèmes où chaque groupe contient 100 instances, et où m est le nombre d'objets et n est le nombre d'offres. Les cinq groupes sont donnés comme suit :
– REL-1000-500 : contenant 100 instances allant de in101 à in200, et où m=500 et n=1000.
– REL-1000-1000 : contenant 100 instances allant de in201 à in300, et où m=1000 et n=1000.
– REL-500-1000 : contenant 100 instances allant de in401 à in500, et où m=1000 et n=500.
– REL-1500-1000 : contenant 100 instances allant de in501 à in600, et où m=1000 et n=1500.

– REL-1500-1500 : contenant 100 instances allant de in601 à in700, et où m=1500 et n=1500.

4.2 Algorithmes de comparaisons

Afin de bien évaluer la qualité des solutions trouvées, nous avons implémenté un algorithme de recherche tabou utilisant le codage RK pour générer la solution de départ, où une liste tabou et une phase de diversification évitant la stagnation du processus de recherche ont été mis en oeuvre.

Dans le but de tester et de prouver l'efficacité de notre approche, une étude comparative avec quelques algorithmes de l'état de l'art concernant le *PDG* à savoir Casanova et SAGII [6] est établie dans cette section.

4.2.1 Casanova

Casanova est une méthode de recherche locale proposée par Hoos et Boutilier [8]. La méthode se résume comme suit : l'algorithme démarre d'une allocation vide ou toutes les offres sont considérées initialement non satisfaites. Ensuite, le processus de recherche sélectionne à chaque itération une offre non satisfaite à inclure dans l'allocation et enlève toute offre incompatible pouvant se produire dans l'allocation courante quand de nouvelles offres sont considérées (deviennent satisfaites). À chaque étape, une offre est sélectionnée comme suit :

1. Avec une probabilité wp ($Walk\ probability$) une offre non satisfaite est sélectionnée aléatoirement.
2. Avec une probabilité $1 - wp$, les offres sont classées selon leur profit qui correspond au prix de l'offre divisé par le nombre d'objets couverts par l'offre. Ensuite et avec une probabilité np ($Novelty\ probability$) la meilleure offre, b_1 ayant le plus grand profit est sélectionnée pour être incluse dans l'allocation courante. Autrement et avec une probabilité $1-np$, la deuxième meilleure offre b_2 est choisie pour inclusion dans l'allocation courante.

Le processus de recherche est réitéré pour un certain nombre d'itérations. Ce processus est similaire à celui d'adaptNovelty, la méthode de recherche locale proposée par le même auteur pour résoudre le problème de la satisfiabilité [9].

4.2.2 SAGII

Dans [6], Guo *et al.* ont proposé une heuristique de recuit simulé combinée avec un algorithme de Branch-and-Bound. Les résultats sont très encourageants et la nouvelle méthode surpasse clairement Casanova. L'heuristique proposée fait appel tout d'abord à un prétraitement pour exclure les offres qui peuvent mener aux solutions suboptimales, ce qui permettra d'améliorer le temps de réponse du processus de recherche.

Le processus de recherche consiste en trois composantes :

1. La première composante est un algorithme Branch-and-Bound appliqué au sous-ensemble d'objets des offres de l'allocation courante.
2. La deuxième composante est une recherche locale utilisée pour sélectionner les meilleures offres à inclure dans l'allocation courante.
3. La troisième composante est un mouvement (échange) qui permet de sélectionner d'une manière aléatoire l'offre à considérer dans l'allocation.

La méthode démarre d'une allocation vide. Ensuite la phase de prétraitement est lancée pour exclure les offres qui peuvent mener aux solutions sub-optimales. Une fonction de pénalité est utilisée pour éliminer les offres incompatibles.

L'algorithme Branch-and-Bound est exécuté avec une probabilité p_1 =0.2, la recherche locale est lancée avec une probabilité p_2=0.7, et le mouvement échange est exécuté avec une probabilité 1-p_1-p_2.

Le processus de recherche qui consiste en un algorithme de Branch-and-Bound, une recherche locale et un mouvement d'échange est répété pour un certain nombre d'itérations.

4.2.3 Recherche tabou

La méthode de recherche tabou que nous proposons démarre d'une solution créée aléatoirement et selon le codage RK. Puis la meilleure allocation voisine est sélectionnée pour être la prochaine solution courante. Pour produire des solutions voisines, nous définissons deux mouvements qui sont : "On-Bid" et "On-Item".

1. Mouvement "on-Bid" : l'espace d'état est défini par l'ensemble des offres insatisfaites qui ne sont pas dans l'allocation courante. Ces offres sont considérées admissibles et peuvent être incluses dans l'allocation courante. La meilleure offre (celle qui maximise le prix global quand on l'ajoute à l'allocation) dans le voisinage actuel est choisie pour être incluse dans l'allocation courante et toutes les offres incompatibles dans l'allocation sont à enlever.
2. Mouvement "on-Item" : L'espace d'état est défini par l'ensemble des objets qui ne sont pas couverts par les offres dans l'allocation courante. La meilleure offre couvrant tels objets est choisie pour être incluse dans l'allocation courante

et toutes les offres incompatibles dans l'allocation sont enlevées.

Après avoir généré toutes les configurations voisines en utilisant l'opérateur de mouvement[4], la meilleure configuration non tabou est sélectionnée pour être l'allocation candidate.

Afin d'échapper aux allocations déjà visitées, notre stratégie de recherche utilise une liste pour maintenir les numéros d'offres récemment choisies. Ces numéros sont dits "mouvements tabous". Quand un mouvement tabou appliqué à une allocation courante donne une meilleure solution ; nous acceptons ce mouvement malgré son statut tabou par critère d'aspiration.

Une étape de diversification est lancée pour explorer de nouvelles régions, ce qui évitera la stagnation du processus de recherche. L'étape de diversification consiste à choisir aléatoirement une offre insatisfaite pour être incluse dans la meilleure allocation courante et toutes les offres incompatibles sont enlevées. La stratégie de diversification est répétée pour n étapes consécutives où n est le nombre d'offres.

La recherche s'arrête après un certain nombre d'itérations fixé d'une manière empirique.

Suite à plusieurs tests, les paramètres de notre recherche tabou (TS) sont fixés comme suit : le nombre d'exécutions (max_trials) est fixé à 500, le nombre maximum d'itérations (max_iter) est fixé à 500, et la taille de la liste tabou est fixée à 40.

4.3 Valeurs des paramètres

L'efficacité d'une approche dépend largement d'un bon ajustement de ses paramètres. Ceux-ci sont fixés suivant des expérimentations effectuées (non reportées ici) et les valeurs prises sont celles pour lesquelles il existe un compromis entre la qualité de la solution obtenue par l'approche et le temps d'exécution de l'algorithme.

L'algorithme SLS développé a été testé sur un certain nombre de problèmes *PDG* difficiles. Suite à plusieurs tests, les paramètres de notre SLS sont fixés comme suit : le nombre maximum d'itérations (max_iter) est fixé à 10000, et la probabilité wp à 0.3. Les résultats correspondant sont donnés dans les tables de 1 à 8.

Les résultats concernant SAGII et Casanova sont tirés du papier [6], où la machine utilisée est un Pentium IV 2.4 Ghz avec 1 GO de RAM, similaire à celui utilisé dans notre étude expérimentale.

[4]Un mouvement consiste à inclure une nouvelle offre dans l'allocation courante.

Tab. 1 – SLS .vs TS sur quelques instances de REL 1000-1000

Ins	*time*	*sol_SLS*	*time*	*sol_TS*
in201	13.90	77499.82	85.37	**81557.74**
in202	14.71	**84283.75**	187.42	82879.72
in203	14.78	**81026.14**	78.37	79505.18
in204	14.70	**81969.04**	140.87	**81969.04**
in205	13.84	**82469.19**	106.73	79977.89
in206	13.95	**86014.73**	124.09	79436.56
in207	14.78	**91033.51**	78.96	**91033.51**
in208	14.84	79546.34	133.76	**82099.07**
in209	14.23	**81328.85**	61.90	80786.38
in210	14.46	**85079.98**	164.53	83350.42
in211	13.43	77348.08	66.20	**79746.14**
in212	14.81	**80944.91**	158.89	80715.77
in213	14.81	79698.25	117.84	**81740.95**
in214	13.78	**81935.32**	80.42	77525.90
in215	14.67	83259.58	120.99	**85549.66**
in216	14.78	78666.80	89.40	**78777.96**
in217	13.17	**81978.09**	137.65	81545.43
in218	14.90	82933.86	88.54	**86936.78**
in219	13.78	**88054.28**	134.93	86064.50
in220	14.85	**86937.85**	124.54	**86937.85**
in290	14.00	**81691.16**	107.48	81053.27
in291	14.29	**87680.27**	51.42	81294.33
in292	14.82	**83794.08**	49.01	82221.24
in293	14.40	**76687.10**	54.64	76017.25
in294	14.28	80080.35	49.98	**80903.21**
in295	14.07	**83086.79**	48.01	**83086.79**
in296	13.64	84013.43	103.74	**84754.59**
in297	13.70	**78667.14**	53.50	78341.52
in298	14.10	**83389.87**	73.39	81659.42
in299	14.93	82295.59	89.25	**84587.35**
in300	14.32	**88373.32**	78.82	83349.51

4.4 Résultats

Les tables de 1 à 5 donnent quelques résultats trouvés par la recherche locale stochastique (SLS) et la recherche tabou (TS) sur quelques instances difficiles du problème PDG. Pour chaque méthode, nous donnons la qualité de solution *sol* et le temps de calcul *time* en seconde nécessaire pour atteindre cette solution. *Ins* est le nom de l'instance PDG.

La table 1 donne les résultats trouvés par TS et SLS sur quelques instances du groupe REL-1000-1000. Les qualités de solutions données par SLS et TS sont comparables alors que SLS est nettement meilleure en temps de calcul.

La table 2 montre les résultats trouvés par TS et SLS sur quelques instances du groupe REL-1500-1000. SLS s'exécute plus vite que TS mais les qualités de

TAB. 2 – SLS .vs TS sur quelques instances de REL 1500-1000

Ins	*time*	*sol_SLS*	*time*	*sol_TS*
in501	15.62	77140.72	98.71	**82216.35**
in502	15.98	**78574.26**	120.82	74127.61
in503	15.99	**79554.65**	114.11	77005.81
in504	16.48	**81903.02**	155.54	**81903.02**
in505	15.53	77836.39	163.84	**77977.65**
in506	12.89	**76722.13**	151.98	74708.78
in507	16.59	**76798.45**	121.93	75916.19
in508	14.48	**79885.98**	341.75	**79885.98**
in509	16.53	**82672.25**	250.32	76115.86
in510	13.15	78411.78	134.21	**78841.15**
in511	13.29	**78793.83**	104.92	**78793.83**
in512	14.43	**81240.92**	188.36	79739.67
in513	14.34	74697.44	189.68	**75425.91**
in514	14.87	**83554.74**	124.50	78416.59
in515	13.29	**81033.82**	560.62	78874.25
in516	15.18	**83663.75**	683.67	**83663.75**
in517	13.45	**72616.07**	139.92	72569.57
in518	15.98	**80027.38**	114.17	**80027.38**
in519	14.87	**80892.13**	109.39	**80892.13**
in520	16.39	**81658.50**	109.34	**81658.50**

solutions trouvées par les deux méthodes sont comparables.

D'après la table 3 représentant les tests effectués sur quelques instances du groupe REL-500-1000, on constate qu'en général SLS donne des solutions qui sont meilleures que celles trouvées par TS.

La table 4 présente les résultats trouvés par TS et SLS sur des instances du groupe 1000-500. Les deux méthodes arrivent à trouver des solutions de qualités comparables. Cependant, TS s'exécute moins vite que SLS.

D'après la table 5 représentant les résultats trouvés par SLS et TS sur quelques instances du groupe REL-1500-1500, on remarque que SLS trouve plus vite des solutions de meilleures qualités alors que TS parfois fournit la même solution mais en un temps moins vite que SLS.

En général, l'algorithme de recherche locale s'est avéré très efficace par rapport à la recherche tabou (TS). Il est souvent meilleur en ce qui concerne la qualité de la solution obtenue et le temps de calcul qui lui est affecté.

TAB. 3 – SLS .vs TS sur quelques instances de REL 500-1000

Ins	*time*	*sol_SLS*	*time*	*sol_TS*
in401	5.67	**72948.07**	44.14	68485.81
in402	5.79	71454.78	23.5780	**72820.03**
in403	6.01	**74843.96**	34.15	**74843.96**
in404	6.12	**78761.68**	16.85	73385.62
in405	6.04	**72674.25**	15.90	**72674.25**
in406	5.87	**71791.03**	37.12	**71791.03**
in407	6.35	**73278.66**	15.57	71578.48
in408	5.95	**72580.04**	27.37	70144.19
in409	5.48	**67177.35**	25.48	**67177.35**
in410	6.37	71791.57	14.01	**72791.68**
in411	5.96	**68548.03**	28.29	**68548.03**
in412	5.78	**75292.63**	13.71	**75292.63**
in413	6.17	**73350.87**	19.34	72561.90
in414	6.42	**71217.93**	40.98	70344.38
in415	5.54	71926.73	14.1720	**78106.73**
in416	6.01	**71211.64**8	75.50	70358.95
in417	6.04	**74680.99**	39.07	69895.95
in418	6.09	**69460.64**	29.70	67272.32
in419	5.84	66810.92	13.3750	**73332.96**
in420	5.62	**71381.02**	23.78	**71381.02**
in490	5.51	66467.12	29.00	**71423.73**
in491	6.20	**74284.64**	21.86	74181.24
in492	6.17	**71142.21**	17.62	70296.63
in493	5.45	**72512.77**	27.59	72197.60
in494	5.98	**72922.96**	14.87	**72922.96**
in495	5.65	71599.56	25.79	**72919.67**
in496	6.26	**73376.74**	30.35	**73376.74**
in497	5.79	**72121.54**	27.64	70942.18
in498	5.39	72844.25	13.20	**74213.25**
in499	6.20	**71246.87**	13.60	**71246.87**
in500	5.70	**69909.96**	23.34	**69909.96**

4.5 Comparaisons avec Casanova, TS et SAGII

La Table 6 (respectivement Table 7) donne les différents résultats trouvés par *SLS* et *Casanova* (respectivement TS) où la colonne μ correspond à la qualité de solution moyenne des 100 instances dans chaque groupe, trouvée par chaque méthode. La colonne *time* donne le temps moyen en seconde nécessaire pour trouver la solution. La colonne δ est le taux d'amélioration qui est égale à $(\mu_{SLS} - \mu_{Casanova})/\mu_{SLS}$ (respectivement $(\mu_{SLS} - \mu_{TS})/\mu_{SLS}$).

La table 8 présente les différents résultats trouvés par SLS et $SAGII$ sur les cinq groupes de problèmes où la colonne δ est le taux d'amélioration qui vaut $(\mu_{SLS} - \mu_{SAGII})/\mu_{SLS}$.

Tab. 4 – SLS .vs TS sur quelques instances de REL 1000-500

Ins	*time*	*sol_SLS*	*time*	*sol_TS*
in101	23.51	**66170.61**	57.86	**66170.61**
in102	23.89	**65466.95**	63.43	64716.31
in103	24.79	**66350.99**	128.68	**66350.99**
in104	22.92	**67268.71**	120.56	62524.23
in105	24.31	**65345.23**	105.36	64312.56
in106	22.37	63479.26	129.42	**64591.70**
in107	23.18	**66245.70**	128.51	63972.62
in108	24.01	**71505.66**	119.84	68776.34
in109	22.20	61751.22	80.98	**64343.07**
in110	23.25	**64083.64**	115.31	60275.66
in111	24.46	**71988.39**	107.87	66546.82
in114	24.14	63664.64	69.37	**69452.16**
in115	23.71	60526.25	82.42	**62579.47**
in116	24.79	63708.48	67.62	**64849.85**
in117	21.39	62249.37	67.65	**64229.21**
in118	21.75	65010.98	105.28	**65843.74**
in119	21.31	60907.91	99.39	**65460.73**

Tab. 5 – SLS .vs TS sur quelques instances de REL 1500-1500

Ins	*time*	*sol_SLS*	*time*	*sol_TS*
in601	15.54	96255.53	100.76	**97473.85**
in602	15.71	**95328.21**	155.34	93873.31
in603	15.48	**94126.96**	137.95	92568.61
in604	15.59	**103568.86**	96.70	92869.78
in605	17.36	**98799.71**	175.14	95787.59
in606	15.60	**104346.07**	334.12	**104346.07**
in607	15.89	**100417.40**	267.79	98674.39
in608	15.26	**95671.77**	95.62	91554.61
in609	16.76	**98566.94**	103.10	96652.44
in610	17.57	**99975.09**	146.03	**99975.09**
in690	16.28	**100613.71**	125.28	**100613.71**
in691	15.56	**102167.67**	96.03	98952.00
in692	17.21	97081.65	95.70	**98353.00**
in693	16.53	**96049.34**	156.81	95603.90
in694	16.23	**108114.12**	339.39	101816.08
in695	17.28	**97687.15**	202.00	**97687.15**
in696	16.26	95908.35	245.28	**96410.53**
in697	15.56	**95890.06**	192.64	91787.14
in698	16.76	**96266.64**	305.68	95572.62
in699	17.26	**103762.70**	187.90	**103762.70**
in700	16.54	**101510.20**	198.65	**101510.20**

A partir de la table 6, on remarque que la méthode de recherche locale stochastique SLS donne une amélioration de 28 à 43 pour cent dans les résultats en comparaison avec Casanova. Les résultats de *SLS* sont nettement meilleurs que ceux fournis par Casanova, concernant la qualité des solutions et le temps mis pour les trouver.

D'après la table 7 représentant les tests effectués sur les classes de problèmes difficiles de PDG, on remarque que les résultats trouvés par la SLS sont meilleurs à ceux donnés par TS. La SLS arrive à trouver des solutions de bonne qualité en un temps de calcul réduit par rapport à la recherche tabou.

A partir de la table 8, on remarque que malgré les outils sophistiqués (Branch-and bound et la phase du prétraitement) utilisés dans *SAGII*, la méthode *SLS* arrive à trouver des solutions comparables à celles de *SAGII* et la différence entre *SLS* et *SAGII* n'est pas très grande.

En général, on peut dire que les résultats trouvés sont très encourageants et montrent l'intérêt de notre approche. La SLS donne des solutions qui sont nettement meilleures que celles trouvées par Casanova et TS. Les résultats trouvés par SAGII sont comparables à ceux donnés par notre approche. Les Figures 1 et 2 le montrent avec clarté.

L'efficacité de notre approche s'explique par la bonne combinaison entre la diversification (explorer de nouvelles régions par l'utilisation de l'aléatoire dans la génération de l'allocation courante et le choix de l'offre à inclure dans l'allocation) et l'intensification (renforcer la recherche dans une zone particulière par le choix de meilleures offres à inclure dans l'allocation) ce qui mène l'algorithme à explorer efficacement l'espace de recherche et localiser une bonne solution.

5 Conclusion

Dans ce papier, nous avons étudié le problème de la détermination du gagnant *PDG* dans les enchères combinatoires. Nous avons proposé une recherche locale stochastique et utilisant le codage *RK* pour trouver une allocation optimale. Plusieurs expérimentations ont été menées sur une grande variété de benchmarks et des comparaisons avec d'autres méthodes telles que *Casanova*, TS et SAGII montrent l'efficacité de notre approche.

Comme perspective, nous souhaitons réaliser dans le futur un système à base d'agents utilisant la recherche locale pour simuler le mécanisme d'enchères combinatoires.

Tab. 6 – SLS vs. Casanova

Test set	*time*	$\mu_{\_SLS}$	*time*	$\mu_{Casanova}$	$\delta\%$
REL-500-1000	**22.35**	**64216.14**	119.46	37053.78	42,30
REL-1000-500	**5.91**	**72206.07**	57.74	51248.79	40,89
REL-1000-1000	**14.19**	**82120.31**	111.42	51990.91	36,68
REL-1000-1500	**14.97**	**79065.08**	168.24	56406.74	28,65
REL-1500-1500	**16.47**	**98877.07**	165.92	65661.03	33,59

Tab. 7 – SLS vs. TS

Test set	*time*	$\mu_{\_SLS}$	*time*	μ_{TS}	$\delta\%$
REL-500-1000	**22.35**	64216.14	91,07	**65286,94**	-1.64
REL-1000-500	**5.91**	**72206.07**	25,84	71985,34	0.30
REL-1000-1000	**14.19**	**82120.31**	104,30	81633,63	0.60
REL-1000-1500	**14.97**	**79065.08**	223,37	77931,41	1.43
REL-1500-1500	**16.47**	**98877.07**	175,68	97824,64	1.06

Tab. 8 – SLS vs. SAGII

Test set	*time*	$\mu_{\_SLS}$	*time*	μ_{SAGII}	$\delta\%$
REL-500-1000	**22.35**	64216.14	38.06	**64922.02**	-1.08
REL-1000-500	**5.91**	72206.07	24.46	**73922.10**	-2.32
REL-1000-1000	**14.19**	82120.31	45.37	**83728.34**	-1.92
REL-1000-1500	**14.97**	79065.08	68.82	**82651.49**	-4.33
REL-1500-1500	**16.47**	98877.07	91.78	**101739.64**	-2.81

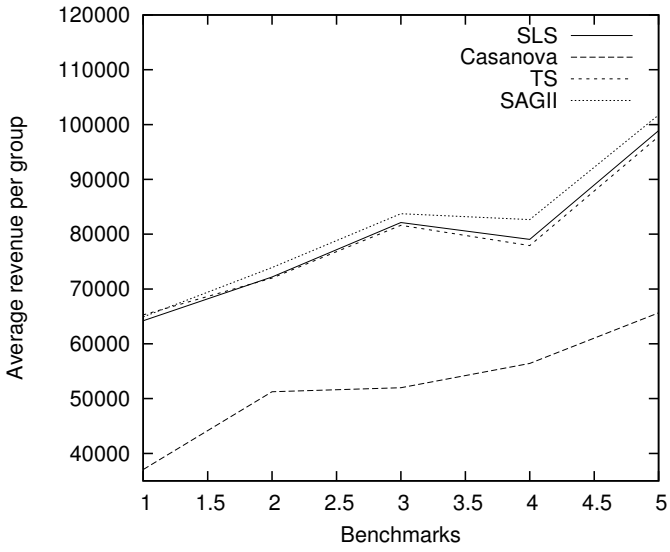

Fig. 1 – Comparaison entre SLS, Casanova, TS et SAGII concernant la qualité de solution.

Références

[1] Anderson, A., Tenhunen, M. and Ygge, F. (2000) 'Integer programming for combinatorial auction winner determination', *In Proceedings of 4th International Conference on Multi-Agent Systems*, IEEE Computer Society Press, July, pp.39-46.

[2] Bean. J.C, (1994). 'Genetics and random keys for sequencing and optimization', *In ORSA Journal of Computing*, Vol 6, nÂ°2, pp 154-160.

[3] Boughaci.D, Benhamou.B, Drias.H, Memetic Algorithms for the Optimal Winner Determination Problems in Combinatorial Auctions, *In Journal of SoftComputing* (to appear).

[4] Fujishima Y, Leyton-Brown K, Shoham Y. (1999) 'Taming the computational complexity of combinatorial auctions : optimal and approximate approaches'. *In Sixteenth international joint conference on artificial intelligence*, pp. 548-553.

[5] Guo Y, Lim A, Rodrigues B, Zhu Y, (2004). 'Heuristics for a brokering set packing problem'. *In Proceedings of eighth international symposium on artificial intelligence and mathematics*, p. 10-14.

[6] Guo Y, Lim A, Rodrigues B, Zhu Y, (2006). 'Heuristics for a bidding problem'. *In Computers and Operations Research* Vol 33, Issue 8 , August 2006, pp : 2179-2188.

[7] Holland A , O'sullivan B. (2004). 'Towards Fast Vickrey Pricing using Constraint Programming', *In Artificial Intelligence Review* , Vol 21, nÂ° 3-4 / June, pp : 335-352.

[8] Hoos H, Boutilier C. (2000), 'Solving combinatorial auctions using stochastic local search'. *In Proceedings of the 17th national conference on artificial intelligence*, pp. 22-29.

[9] Hoos. H. (2002). 'An Adaptive Noise Mechanism for WalkSAT'. *AAAI/IAAI 2002*. pp. 655-660.

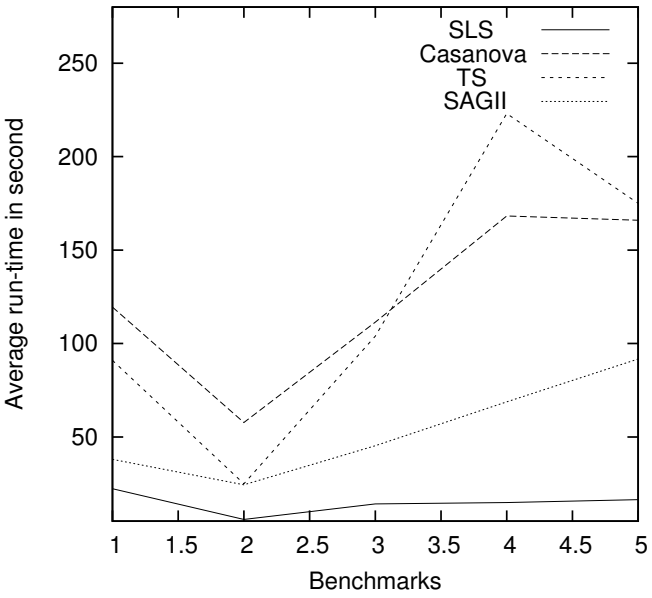

FIG. 2 – Comparaison entre SLS, casanova, TS and SAGII concernant le temps de calcul.

[10] Lau HC, Goh, YG. 'An intelligent brokering system to support multi-agent web-based 4th-party logistics'. *In Proceedings of the 14th international conference on tools with artificial intelligence*, 2002, p. 154-161.

[11] Leyton-Brown K, Tennenholtz M, Shoham Y (2000). 'An Algorithm for Multi-Unit Combinatorial Auctions'. *In Proceedings of the 17th national conference on artificial intelligence* , Austin, Games-2000, Bilbao, and ISMP-2000, Atlanta.

[12] McAfee, R. and McMillan, P.J. (1987) 'Auctions and bidding', *In Journal of Economic Literature*, Vol. 25, pp.699-738.

[13] Nisan, N. (2000) 'Bidding and allocation in combinatorial auctions', *In Proceedings of the ACM Conference on Electronic Commerce* (EC-00), Minneapolis : ACM SIGecom, ACM Press, October, pp.1-12. 21.

[14] Rothkopf, M.H., Pekee, A. and Ronald, M. (1998) 'Computationally manageable combinatorial auctions', *In Management Science*, Vol. 44, No. 8, pp.1131-1147.

[15] Sandholm T, Suri S, Gilpin A, Levine D. (2001), CABoB : 'a fast optimal algorithm for combinatorial auctions'. *In Proceedings of the International joint conferences on artificial intelligence*, p. 1102-1108. 25.

[16] Sandholm T, Suri S, (2000). 'Improved Optimal Algorithm for Combinatorial Auctions and Generalizations'. *In Proceedings of the 17th national conference on artificial intelligence*, pp. 90-97.

[17] Sandholm T, (2006). 'Optimal Winner Determination Algorithms' . *In P. Cramton et al. . (ed.), Combinatorial Auctions,* MIT Press.

[18] Vries de S and R. Vohra,(2003), 'Combinatorial auctions a survey', *In INFORMS Journal of Computing*, Vol 15, pp. 284-309.