Un algorithme hybride pour le problème de sac à dos multi-objectifs

Vincent Barichard et Jin-Kao Hao
LERIA - Faculté des Sciences - Université d'Angers
2 Boulevard Lavoisier
49045 Angers Cedex 01
email : {Vincent.Barichard, Jin-Kao.Hao}@info.univ-angers.fr

Résumé

Nous présentons un algorithme hybride pour le problème de sac à dos multi-objectifs multidimensionnel en 0-1. Cet algorithme, appelé GTS^{MOKP}, combine une procédure génétique avec un opérateur de recherche tabou. GTS^{MOKP} a été testé sur 9 instances connues et a montré des résultats très compétitifs par rapport à deux algorithmes de l'état de l'art.

1 Introduction

Soit un ensemble d'objets, à chacun étant associé un vecteur de gains et de poids. Le problème de sac à dos multi-objectifs multidimensionnel (MOKP) consiste à sélectionner un sous-ensemble d'objets maximisant une fonction multi-objectifs tout en satisfaisant un ensemble de contraintes. Plus formellement, le MOKP peut être défini comme suit :

$$\text{MOKP01}\begin{cases} \max & z^j(x) = \sum_{i=1}^{n} c_i^j x_i & j = 1,...,o \\ \text{s.t.} & \sum_{i=1}^{n} w_i^l x_i \leq b_l & l = 1,...,m \\ & x_i \in \{0,1\} & i = 1,...,n \end{cases}$$

où n est le nombre d'objets, x_i une variable de décision, o le nombre d'objectifs, z^j la $j^{ème}$ composante de la fonction multi-objectifs z, et m le nombre de contraintes du problème.

Comme le problème NP-difficile de sac à dos multidimensionnel (MKP), le MOKP peut être utilisé pour modéliser de nombreux problèmes réels comme la répartition de budgets et l'allocation de ressources. Plusieurs algorithmes heuristiques ont été développés pour résoudre le MOKP. Des méthodes de voisinage comme le recuit simulé [10, 12] et la recherche tabou [4], des méthodes évolutionnaires : "vector evaluated genetic algorithm" (VEGA) [9], "non-dominated sorting genetic algorithm" (NSGA) [11], "strength Pareto evolutionary algorithm" (SPEA) [14], et "memetic Pareto archived evolution strategy algorithm" (M-PAES) [1]. Très récemment, des algorithmes hybrides combinant la recherche génétique et la recherche locale (limitée à des méthodes de descentes à notre connaissance)[1] tels que "multiple objective genetic local search" (MOGLS) [6] ainsi que des approches hybrides parallèles [8] ont été proposés.

Parmi toutes les approches proposées, les meilleurs résultats observés sur les instances que nous allons utiliser sont détenus par l'algorithme MOGLS présenté dans [6].

Dans ce papier, nous nous intéressons à l'approche hybride et présentons GTS^{MOKP}, un algorithme génétique combiné à une recherche tabou pour le MOKP. Inspiré par un algorithme hybride pour le problème de coloriage de graphe [3], et par l'algorithme MOGLS [6], GTS^{MOKP} se distingue de MOGLS essentiellement par l'intégration d'une méthode de recherche de voisinage très performante (la méthode tabou) au schéma génétique. Pour évaluer les performances de GTS^{MOKP}, l'algorithme est testé sur un jeu de 9 instances publiées de MOKP, puis comparé avec NSGA et MOGLS, deux algorithmes reconnus pour résoudre le MOKP. Les résultats expérimentaux obtenus par GTS^{MOKP} se sont montrés très compétitifs sur toutes les instances comparativement aux algorithmes en compétition.

Le papier est organisé comme suit. Dans la prochaine section, nous présentons les principes généraux de notre algorithme hybride GTS^{MOKP}. Les résultats expérimentaux ainsi que les comparaisons de GTS^{MOKP} avec les deux autres algorithmes testés sont le sujet de la section 3. Dans la dernière section nous donnons quelques conclusions et perspectives.

2 Génétique Tabu Search pour le MOKP (GTS^{MOKP})

L'hybridation entre une recherche génétique et une recherche locale est maintenant reconnue comme une approche compétitive pour attaquer des problèmes combinatoires difficiles. Pour le MOKP, nous pouvons citer l'algorithme MOGLS utilisant une méthode de descente pure comme opérateur de recherche locale [6].

Les principes généraux pour hybrider un algorithme génétique avec une méthode de recherche locale sont connus et devenus presque des standards pour ce type d'hybridation. À chaque génération, un nombre de croisements (avec un opérateur de croisement spécifique au problème ou aléatoire) est effectué. Pour chaque nouvelle configuration (individu) s ainsi générée, un opérateur de recherche locale $LS(s)$ est appliqué à s pour améliorer sa qualité. Finalement, un mécanisme de sélection décide si le nouvel individu amélioré doit être introduit dans la population.

Dans cette section, nous présentons notre algorithme hybride GTS^{MOKP} pour le MOKP.

1. Le terme de recherche locale est souvent utilisé comme synonyme de méthodes de descentes pures. Dans ce papier, ce terme est utilisé pour désigner toutes les méthodes utilisant le concept de voisinage comme la recherche tabou et le recuit simulé

2.1 Algorithme général

GTS^{MOKP} suit le schéma classique utilisé pour l'hybridation entre algorithme génétique et recherche locale. Nous donnons le squelette de GTS^{MOKP} à l'algorithme 1. Plus de détails sont fournis dans la section 2.2.

Require: une instance MOKP
Ensure: un ensemble de solutions non dominées
  $P \leftarrow$ InitPopulation ($|P|$)
  while Critère d'arrêt non rencontré **do**
    $\lambda \leftarrow$ GetRandomWeightVector (utilisé pour l'évaluation, voir prochaine section)
    Evaluer chaque individu s dans P selon $f(s,r,\lambda)$ (voir prochaine section)
    $TP \leftarrow$ les N "meilleurs" individus de P
    $(p_1,p_2) \leftarrow$ ChooseParents(TP)
    $s \leftarrow$ Crossover(p_1,p_2)
    $s \leftarrow$ TabuSearch(s,$NumberOfIterations$)
    $P \leftarrow$ AddPopulation(s,P)
  end while

Algorithm 1: GTS^{MOKP} : *"genetic tabu search algorithm"* pour le MOKP.

Remarquons toutefois, que bien que non clairement explicitées dans l'algorithme, les solutions non-dominées sont enregistrées dans une structure de données appropriée. C'est cet ensemble de solutions qui est donné en sortie de l'algorithme. Nous présentons dans la prochaine section les différents composants de GTS^{MOKP}.

2.2 Espace de recherche et valeurs d'évaluations

une *configuration* (ou un *individu*) s est un vecteur binaire de n composantes satisfaisant toutes les m contraintes du problème. *L'espace de recherche* s est alors constitué par l'ensemble de tous les vecteurs binaires, qui est clairement un sous-ensemble de $\{0,1\}^n$.

Pour évaluer une configuration s, nous utilisons comme dans [7], une fonction d'agrégation linéaire pondérée, définie comme suit :

$$f(s,r,\lambda) = \sum_{i=1}^{o} \lambda_i * (r_i - z_i(s)) \qquad (1)$$

où

- s la configuration à évaluer;
- $z_i(s)$ (i=1...o) la valeur de s pour la $i^{ème}$ composante de la fonction objective du problème;
- λ un vecteur de poids composé de o composantes comprises entre [0..1], dont la $i^{ème}$ composante λ_i correspond au poids associé au $i^{ème}$ objectif;
- r un vecteur de référence à o-composantes. Ce point de référence est dans notre cas calculé en fonction des valeurs maximales, sur chaque objectif, des points de l'ensemble courant des solutions non dominées : $r_i = \max_{x \in Set}(x_i)$ $i \in [1..o]$.

Ainsi, pour chaque configuration s et pour un vecteur de référence r donné, une valeur d'adaptation ("fitness") est attribuée à s selon l'évaluation et le poids associé à chaque objectif de s. Cette fonction d'évaluation (1) définie un ordre total pour les configurations de l'espace de recherche S. Cet ordre est utilisé comme base pour les opérateurs génétiques et ceux de recherche locale.

Notre fonction d'évaluation étant basée sur une fonction d'agrégation linéaire, certaines solutions Pareto optimales peuvent ne jamais être atteintes. Notons toutefois que grâce au hasard de la recherche heuristique, certaines de ces solutions peuvent quand même être déterminées par notre algorithme.

D'autres fonctions d'évaluations sont aussi possibles comme celles basées sur le "ranking" des solutions, l'ordre lexicographique des objectifs, etc. (voir [2]).

2.3 La recherche génétique

Après avoir assigné une valeur d'adaptation à chaque individu de P, la population courante est triée selon ces valeurs d'adaptation. Les N premiers (meilleurs) individus [2] sont dupliqués dans une population temporaire. Puis, deux individus (de cette population temporaire) sont sélectionnés de manière aléatoire pour être recombinés à travers un opérateur de croisement. Nous utilisons ici le croisement aléatoire mono-point. Après avoir restauré la faisabilité du nouvel individu obtenu, celui-ci est amélioré par une recherche tabou (voir prochaine section). Si l'individu amélioré s_* est meilleur que le plus mauvais individu de la population temporaire, s_* est ajouté à la population courante et remplace le plus vieil individu stocké dans celle-ci. Dans les autres cas, s_* est rejeté.

2.4 La recherche Tabou (TS)

L'opérateur TS *TabuSearch(s,L)* a pour but d'améliorer une configuration réalisable s produite par l'opérateur de croisement, pour un nombre maximum d'itérations L. Cette recherche est effectuée avant d'insérer le s amélioré dans la population.

Cette section rappelle les aspects les plus importants de la recherche Tabou. Pour une présentation approfondie de TS, le lecteur est invité à consulter le livre de Glover et Laguna [5]. Nous présentons ci-dessous quelque notations nécessaires pour une bonne compréhension de notre algorithme TS.

Voisinage et mouvement. La fonction de voisinage $\mathcal{N}$ est définie sur l'espace de recherche S, et associe à chaque configuration s un sous-ensemble de S. Plus précisément, prenons une configuration s, alors $\mathcal{N}=\{s' \mid s'$ est réalisable et Hammingdistance$(s,s')= 1\}$. En d'autres termes, pour chaque $s \in S$, une configuration s' voisine est obtenue par l'ajout (changement d'une variable de 0 à 1) ou par la suppression (changement d'une variable de 1 à 0) d'un objet de s de telle manière que les contraintes restent toujours satisfaites. L'opération de changer une variable pour obtenir une configuration voisine est appelée un *mouvement*. Un mouvement de s vers $s' \in \mathcal{N}$ peut être identifié sans ambiguïté grâce à l'attribut j si s' est obtenu par changement du $j^{ème}$ composant de s.

2. $N \leq |P|$ est à fixer de manière empirique. Dans ce papier, $N = 20$ pour $|P|$ variant de 150 à 350

Évaluation du voisinage. La qualité de chaque configuration voisine $s' \in \mathcal{N}$ est calculée grâce à la fonction d'agrégation linéaire pondérée (1), de la même manière que pour n'importe quelle autre configuration de l'espace de recherche S.

La liste Tabou. Le rôle de la liste de tabou est d'empêcher l'algorithme de tomber dans des cycles à court terme. Comme un mouvement correspond au changement d'une seule variable, l'index de la variable changée, c'est à dire j, est marqué comme tabou pour les k prochaines itérations. La valeur k (taille de la liste tabou) est un paramètre à fixer de manière empirique. Notre taille de liste tabou est fixée à 2 pour chaque exécution de tabou.

Le squelette de notre algorithme TS utilisé dans GTS^{MOKP} est donné à l'algorithme 2.

Require: une configuration réalisable s_0, le nombre d'itérations L
Ensure: une nouvelle configuration réalisable s
  $s \leftarrow s_0$
  for $i = 0$ to L **do**
    choisir le meilleur mouvement autorisé
    mettre à jour la liste tabou
    effectuer le mouvement sélectionné dans s
    mettre à jour l'ensemble des solutions non-dominées avec s
  end for

Algorithm 2: L'algorithme de recherche Tabou.

3 Résultats expérimentaux

3.1 Jeux de données

Dans cette section, nous comparons notre algorithme GTS^{MOKP} avec NSGA et MO-GLS, deux des méthodes les plus performantes pour le MOKP. Les expérimentations sont effectuées sur les 9 instances publiées dans [16]. Ces benchmarks ont 2, 3 et 4 objectifs combinés avec 250, 500 et 750 objets. De plus, pour chaque instance, le nombre de contraintes est égal au nombre d'objectifs. Ces instances ont été générées de manière aléatoire avec des poids et des profits décorélés, et les capacités de chaque contraintes sont réglées à environ la moitié de la somme des poids de la contrainte considérée. Il en découle que le nombre d'objets présents attendu dans les solutions optimales est d'environ la moitié du nombre total d'objets du problème.

3.2 Mesure des performances

Pour évaluer les résultats (le front des meilleurs compromis) produits par les différents algorithmes, nous utilisons deux mesures qui ne dépendent pas de l'échelle de calcul de chaque objectif : *La taille de l'espace non dominé* ($\mathcal{S}$) et *la couverture de deux ensembles* [15].

Définition 1 *La taille de l'espace non dominé* ($\mathcal{S}$). *Soit* $A = (x_1,x_2,...,x_l) \subseteq X$ *un sous-ensemble de* l *vecteurs de décision. La fonction* $\mathcal{S}$ *calcule le volume borné par l'union de tous les polytopes* $p_1,p_2,...,p_l$, *où chaque* p_i *est formé par l'intersection des hyperplans des* x_i *par rapport aux axes du repère : pour chaque axe de l'espace des objectifs, il existe un hyperplan perpendiculaire à l'axe et passant par le point* $(f_1(x_i),f_2(x_i),...,f_k(x_i))$. *Pour le cas à deux dimensions, chaque* p_i *représente un rectangle défini par les points de coordonnées* $(0,0)$ *et* $(f_1(x_i),f_2(x_i))$.

Cette mesure ne peut pas être utilisée pour comparer deux ensembles relativement l'un par rapport à l'autre. Pour calculer un ratio permettant de comparer deux ensembles, nous appliquons une deuxième mesure.

Définition 2 *La couverture de deux ensembles. Soit* $A,B \subseteq X$ *deux ensembles de vecteurs de décisions. La fonction* $\mathcal{C}$ *associe à chaque pair ordonnée* (A,B) *une valeur de l'intervalle* $[0,1]$ *:*

$$\mathcal{C}(A,B) := \frac{|\{b \in B \mid \exists\, a \in A : a \geq b\}|}{|B|}$$

La valeur $\mathcal{C}(A,B) = 1$ *signifie que tous les vecteurs de décisions de* B *sont faiblement dominés*[3] *par ceux de* A. *A l'inverse,* $\mathcal{C}(A,B) = 0$, *signifie qu'aucun des points de* B *n'est faiblement dominé par un point de* A.

3.3 Paramètres expérimentaux

L'algorithme GTS^{MOKP} est programmé en CAML et compilé avec OCAML. Pour obtenir des comparaisons équitables avec les autres algorithmes de l'état de l'art, nous avons aussi implémenté, toujours en CAML, NSGA et MOGLS, deux algorithmes reconnus comme très performants pour le MOKP. Dans notre implémentation, les principales structures de données sont partagées par les trois algorithmes. Cela nous permet d'avoir une base solide pour des comparaisons équitables entre ces algorithmes.

Avant de comparer GTS^{MOKP}, NSGA et MOGLS, nous avons d'abord comparé notre implémentation de NSGA et MOGLS avec celles de MOMHLib [7]. Les résultats obtenus nous permettent d'affirmer que les deux implémentations ont des performances comparables.

Dans nos expérimentations, nous utilisons les paramètres suivants :

- pour NSGA, nous initialisons le taux de mutation à 0.2, et la distance de voisinage à 0.4, en accord avec [16] et [6];
- pour GTS^{MOKP}, nous initialisons le nombre de générations à 50, le nombre d'itérations tabou à $L = 12$ et la taille de la liste tabou à 2 pour chaque exécution de l'algorithme tabou;
- pour NSGA et MOGLS, nous autorisons toujours un nombre largement supérieur de générations pour obtenir des temps de calcul supérieurs ou égaux à ceux obtenus pour GTS^{MOKP} (voir tableau 1);

3. Un point u domine faiblement un point v si et seulement si, $\forall i \in \{1,2,...,k\} : u_i \geq v_i$.

Instances		Méthodes			
Nombre d'objectifs	Nombre d'objets	Taille de la population (tous les algorithmes)	Nombre de générations		
			NSGA	MOGLS	GTS^{MOKP}
2	250	150	500	110	50
	500	200	500	140	50
	750	250	500	150	50
3	250	200	1100	160	50
	500	250	1100	170	50
	750	300	1100	170	50
4	250	250	3000	300	50
	500	300	3000	320	50
	750	350	3000	320	50

TAB. 1 – *Valeurs des paramètres pour les différents algorithmes et instances.*

- L'initialisation de la taille de la population dépend du nombre d'objets ainsi que du nombre d'objectifs de l'instance traitée. Sur une même instance, la taille de population est la même pour les trois algorithmes (voir tableau 1).

Notons que pour MOGLS et GTS^{MOKP}, le critère d'arrêt est calculé comme suit :

$$\text{critère d'arrêt} = \text{Nombre de générations} * \text{taille de la population}.$$

La notion de génération n'est présente que pour garder la présentation classique des paramètres des algorithmes évolutionnaires.

Le tableau 1 récapitule les valeurs des principaux paramètres utilisés par NSGA, MOGLS et GTS^{MOKP}.

Le tableau 2 montre pour chacun des trois algorithmes comparés et pour chaque instance du problème, le temps de calcul obtenu pour les valeurs des paramètres[4]. Notons que NSGA demande toujours plus de temps de calcul que MOGLS, qui lui même demande plus de temps de calcul que GTS^{MOKP}. Donc, nous sommes sûrs que GTS^{MOKP} n'est pas favorisé par rapport aux autres algorithmes en compétition.

3.4 Résultats comparatifs

Dans cette section, nous présentons les résultats expérimentaux obtenus par GTS^{MOKP} ainsi que ceux de NSGA et MOGLS[5]. Les résultats suivants représentent pour chaque instance les moyennes sur plusieurs exécutions indépendantes.

Le tableau 3 montre les résultats obtenus pour la mesure $\mathcal{S}$ (taille de l'espace non dominé). Sur ce tableau, nous observons premièrement que GTS^{MOKP} et MOGLS donnent tout les deux de meilleurs résultats (des valeurs de $\mathcal{S}$ plus élevées) que NSGA sur toutes les instances. En comparant GTS^{MOKP} et MOGLS, nous observons que GTS^{MOKP} dépasse MOGLS pour 7 des 9 instances et donnent des résultats identiques sur les 2 instances bi-objectives restantes (500 et 750 objets). Nous remarquons aussi que GTS^{MOKP} obtient de très bons résultats sur les instances les plus difficiles.

4. Le temps de calcul est basé sur le code généré par le compilateur OCAML sur un PC sous Linux (Bi-Pentium III 1 Ghz).

5. MOGLS détient à notre connaissance les meilleurs résultats expérimentaux sur les instances utilisées (voir [6]).

Nombre d'objectifs	Nombre d'objets	NSGA	MOGLS	GTS^{MOKP}
2	250	60	54	50
	500	205	150	135
	750	330	200	185
3	250	265	250	210
	500	900	490	465
	750	1800	740	660
4	250	1417	1264	1188
	500	3500	2300	2200
	750	7000	3100	2500

TAB. 2 – *Moyenne des temps de calcul pour chaque algorithme (secondes).*

Nombre d'objectifs	Nombre d'objets	NSGA	MOGLS	GTS^{MOKP}
2	250	9.52e+7	9.86e+7	9.87e+7
	500	3.91e+8	4.08e+8	4.08e+8
	750	8.36e+8	8.93e+8	8.93e+8
3	250	8.45e+11	9.33e+11	9.35e+11
	500	6.74e+12	7.72e+12	7.73e+12
	750	2.33e+13	2.71e+13	2.72e+13
4	250	6.97e+15	8.11e+15	8.12e+15
	500	1.09e+17	1.35e+17	1.36e+17
	750	5.52e+17	7.19e+17	7.20e+17

TAB. 3 – *Moyenne de la taille de l'espace non dominé $\mathcal{S}$.*

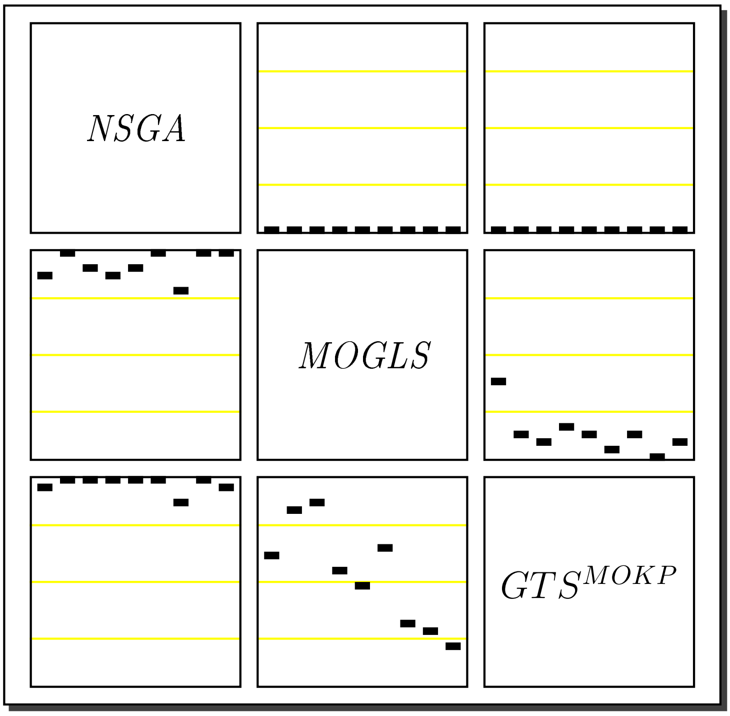

Fig. 1 – *Résultats des comparaisons obtenues avec la mesure $\mathcal{C}$. Chaque rectangle contient neuf petites barres noires représentant la distribution des valeurs de $\mathcal{C}$ pour un ordre particulier de la paire d'algorithmes ; les trois barres noires de gauche correspondent aux instances bi-objectives respectivement (de gauche à droite) de 250, 500 et 750 objets ; les 3 barres noires du milieu correspondent aux instances à 3 objectifs et celles de droite aux instances à 4 objectifs. Sur chaque rectangle, l'échelle va de* 0 *en bas à* 1 *en haut. De plus, à chaque rectangle (l'algorithme A en ligne, et l'algorithme B en colonne) est associée la fraction du nombre d'individus de B faiblement dominés par A :* $\mathcal{C}(A,B)$.

La figure 1 récapitule les résultats de la mesure $\mathcal{C}$ (couverture de deux ensembles) suivant la présentation adoptée dans [16]. Sur cette figure, chaque petite barre noire représente les résultats de la mesure $\mathcal{C}$ entre deux méthodes, pour chaque instance de problème. Du fait d'une très faible dispersion des résultats, la moyenne est pertinente comme valeur retenue pour la mesure $\mathcal{C}$.

Il est observé qu'une fois de plus, les résultats obtenus par NSGA sont inférieurs à ceux de MOGLS et de GTS^{MOKP} pour toutes les instances. En comparant MOGLS et GTS^{MOKP}, nous observons que les résultats penchent en faveur de GTS^{MOKP} sur toutes les instances. En effet, les valeurs obtenues $\mathcal{C}(MOGLS{,}GTS^{MOKP})$ sont inférieures à celles de $\mathcal{C}(GTS^{MOKP}{,}MOGLS)$. Notons que sur plusieurs instances (500 objets 3 objectifs, et 500 objets 4 objectifs) le rapport entre les résultats est supérieur à 10, ce qui signifie que la quasi-totalité des solutions trouvées par GTS^{MOKP} sont de qualité égale ou supérieure à celles trouvées par MOGLS.

En résumé, même si GTS^{MOKP} est exécuté dans des conditions défavorables (moins de générations et moins de temps de calcul autorisé), il donne des résultats au moins égaux ou supérieurs à ceux obtenus par NSGA et MOGLS sur les 9 instances testées avec les deux mesures utilisées. Pour nous convaincre de cette supériorité, nous avons aussi compté pour chaque algorithme le nombre de solutions trouvées qui sont membre du front Pareto optimal. Du fait de la taille des instances du problème considéré, nous n'avons les fronts Pareto optimaux que pour les instances bi-objectives de 250 et 500 objets.

Une fois encore, GTS^{MOKP} obtient de meilleurs résultats comparativement aux autres algorithmes NSGA et MOGLS. En effet, pour la première instance (250 objets), 9.33% des solutions trouvées par GTS^{MOKP} sont des éléments du front Pareto alors que ce taux n'est respectivement que de 0.18% pour NSGA et 6.51% pour MOGLS. Pour la seconde instance (500 objets), ce taux est respectivement de 0.35% pour GTS^{MOKP}, 0.07% pour MOGLS et 0% pour NSGA.

De plus, nous avons observé que dans les autres expérimentations que nous avons effectuées, où le nombre de générations pour GTS^{MOKP} est fixé à la même valeur que celui de MOGLS, les résultats obtenus par GTS^{MOKP} sont encore meilleurs que ceux présentés dans ce papier.

4 Conclusion

Dans ce papier, nous avons présenté GTS^{MOKP}, un algorithme génétique avec recherche tabou très performant pour résoudre le sac à dos multi-objectifs multidimensionnel en 0-1 (MOKP01). Notre algorithme GTS^{MOKP} combine les concepts généraux des méthodes évolutionnaires avec les propriétés locales de la recherche tabou, conduisant à un meilleur compromis entre exploitation et exploration. Les performances de GTS^{MOKP} ont été validées sur un ensemble de 9 instances publiées et comparées avec NSGA et MOGLS, deux algorithmes de l'état de l'art. Les résultats expérimentaux ont montré que GTS^{MOKP}, avec moins d'effort, obtient des meilleurs résultats que les autres algorithmes en compétition sur les instances testées.

Les parties génétique et recherche tabou utilisées dans GTS^{MOKP} sont implémentées de manière naturelle en se basant sur les principaux concepts de ces deux paradigmes. Pour cette raison, nous sommes fortement convaincus que beaucoup d'améliorations sont possibles. Par exemple, sur la partie génétique, d'autres stratégies de sélection peuvent être expérimentées, et des opérateurs de croisements spécifiques au problème peuvent être développés. Pour l'opérateur TS, nous pensons que les techniques développées pour le problème de sac à dos multidimensionnel classique, comme celles présentées dans [13], peuvent apporter beaucoup à un problème du type MOKP.

Remerciements

Nous tenons à remercier les différents relecteurs de ce papier pour leurs commentaires très utiles.

Références

[1] D.W. Corne, J. D. Knowles *M-PAES : a memetic algorithm for multiobjective optimization.* Proceedings of the 2000 congres on evolutionary computation (CEC 2000).

[2] M. Ehrgott, X. Gandibleux *A Survey and annotated Bibliography of Multiobjective Combinatorial Optimization.* OR Spektrum, 22 : 425–460, 2000.

[3] P. Galinier, J.K. Hao *Hybrid evolutionary algorithms for graph coloring.* Journal of Combinatorial Optimization, 3(4) : 379–397, 1999.

[4] X. Gandibleux, N. Mezdaoui, A. Freville *A multiobjective Tabu Search procedure to solve combinatorial optimization problems.* Lecture Notes in Economics and Mathematical Systems, 455 : 291–300, Springer, 1997.

[5] F. Glover, M. Laguna *Tabu Search, Kluwer Academic Publishers.* 1997.

[6] A. Jaszkiewicz *On the performance of multiple objective genetic local search on the 0/1 knapsack problem : a comparative experiment.* Research report, Institute of Computing Science, Poznan University of Technology, RA-002, 2000.

[7] A. Jaszkiewicz *Experiments done with the MOMHLib.* http://www-idss.cs.put.poznan.pl/ jaszkiewicz/MOMHLib/

[8] N. Jozefowiez, F. Semet, E-G. Talbi *Une méta-heuristique parallèle et hybride pour un problème d'élaboration de tournées de véhécules bi-critère.* Quatrième Congrès de la Société Francaise de Recherche Opérationnelle et d'Aide a la décision ROADEF'2002, Paris, 2002.

[9] J.D. Schaffer *Multiple objective optimization with vector evaluated genetic algorithms.* Ph. D. thesis. Vanderbilt University, Unpublished, 1984.

[10] P. Serafini *Simulated annealing for multiobjective optimization problems.* Proc. of 10th Int. Conf. on MCDM, 1 : 87–96, 1992.

[11] N. Srinivas, K. Deb *Multiobjective optimization using non dominated sorting in genetic algorithms.* Evolutionary Computation, 2(3) : 221–248, 1994.

[12] E.L. Ulungu, J. Teghem, Ph. Fortemps, D. Tuyttens *MOSA method : a tool for solving multiobjective combinatorial optimization problems.* Journal of Multi-Criteria Decision Analysis, 8 : 221–336, 1999.

[13] M. Vasquez, J.K. Hao *A hybrid approach for the 0-1 multidimensional knapsack problem.* Proc. of the 13^{th} Intl. Joint Conference on Artificial Intelligence (IJCAI-01). 1 : 328–333, 2001.

[14] E. Zitzler, L. Thiele *An evolutionary algorithm for multiobjective optimization : the stength Pareto approach.* Technical Report 43, 1998.

[15] E. Zitzler, L. Thiele *Multiobjective optimization using evolutionary algorithms : a comparative case study.* Lecture Notes in Computer Science, 1498 : 292–301, Springer, 1998.

[16] E. Zitzler, L. Thiele *Multiobjective evolutionary algorithms : a comparative case study and the strength Pareto approach.* IEEE Transactions on Evolutionary Computation, 3 : 257–271, 1999.