Article invité

Optimisation dynamique : état de l'art et proposition d'une métaheuristique multi-agents

Julien Lepagnot, Amir Nakib, Hamouche Oulhadj et Patrick Siarry [1]
julien.lepagnot@univ-paris12.fr

1 Introduction

Beaucoup de problèmes réels d'optimisation sont variables dans le temps ou dynamiques. De ce fait, un intérêt croissant a été porté à ce type de problèmes, et donc aux algorithmes destinés à les résoudre. Un problème d'optimisation dynamique est caractérisé par une fonction objectif qui change en fonction du temps. C'est le cas, par exemple, de problèmes classiques tels que le problème de tournées de véhicules (VRP) ou le problème d'affectation de tâches.

Le VRP consiste à déterminer les tournées d'une flotte de véhicules afin de livrer un ensemble de clients, tout en minimisant le coût de livraison des biens. Une variante dynamique de ce problème est illustrée dans la Figure 1, où les tournées de différents véhicules, depuis un dépôt central, sont représentées par des cycles dont les sommets correspondent à des clients. La nature dynamique de cette variante vient du fait que des clients peuvent être ajoutés ou supprimés inopinément (problème connu dans la littérature sous l'acronyme DVRP).

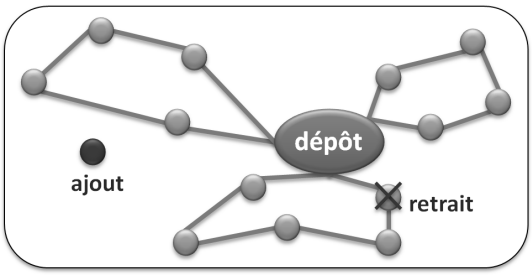

Figure 1 – Le problème de tournées de véhicules.

Une variante dynamique du problème d'affectation de tâches est également illustrée en Figure 2. Il s'agit de répartir un ensemble de tâches sur différentes machines, de manière à minimiser le temps nécessaire pour toutes les traiter. La nature dynamique de ce problème vient du fait que des tâches peuvent apparaître ou disparaître à tout moment, et que des machines peuvent tomber en panne.

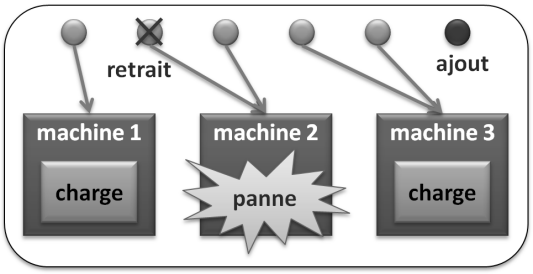

Figure 2 – Le problème d'affectation de tâches.

Une formulation mathématique est donnée en (1), où $f(\vec{x},t)$ désigne la fonction objectif d'un problème de minimisation, $h_j(\vec{x},t)$ désigne la $j^{\text{ème}}$ contrainte d'égalité et $g_k(\vec{x},t)$ la $k^{\text{ème}}$ contrainte d'inégalité. Toutes ces fonctions peuvent varier au cours du temps. Le but n'est pas alors seulement de trouver l'optimum global, mais de le suivre aussi fidèlement que possible dans le temps.

$$\begin{array}{ll} \min & f(\vec{x},t) \\ \text{s.c.} & h_j(\vec{x},t) = 0 \text{ for } j = 1,2,...,u \\ & g_k(\vec{x},t) \leq 0 \text{ for } k = 1,2,...,v \end{array} \quad (1)$$

Le moyen le plus simple pour résoudre un problème dynamique est de redémarrer un algorithme d'optimisation statique (c'est-à-dire un algorithme conçu pour des problèmes dont la fonction objectif n'évolue pas au cours du temps) à chaque fois qu'un changement se produit dans la fonction objectif. Cela nécessite, toutefois, de pouvoir déterminer l'instant du changement et d'avoir suffisamment de temps entre chaque changement pour trouver une solution acceptable.

Une accélération de la convergence peut naturellement être envisagée en utilisant des informations issues des exécutions précédentes de l'algorithme. Par exemple, notons $\vec{o}$ et $\vec{o'}$ les positions de l'optimum global, respectivement avant et après un changement dans la fonction objectif. Si nous supposons que $\vec{o'}$ est "proche" de $\vec{o}$, il peut alors être avantageux de redémarrer la recherche à partir de $\vec{o}$, ou de

1. Laboratoire Images, Signaux et Systèmes Intelligents, LiSSi (E.A. 3956)
Université Paris-Est Créteil, 61, avenue du Général de Gaulle, 94010 Créteil, France

la restreindre au voisinage de $\vec{o}$. Cependant, l'utilisation d'informations sur les précédents états de la fonction objectif ne peut être bénéfique que si l'intensité des changements n'est pas significative. Si les changements sont drastiques, et la fonction objectif complètement transformée après un changement, alors un simple redémarrage de l'algorithme reste la meilleure solution. Néanmoins, pour la plupart des problèmes réels, la fonction objectif ne subit pas de transformations importantes lorsque survient un changement. La question est alors de savoir quelles informations exploiter depuis l'historique de la fonction, et comment les utiliser pour accélérer la convergence.

En pratique, redémarrer un algorithme d'optimisation statique dès qu'un changement a lieu dans la fonction objectif n'est pas toujours possible et peut s'avérer inefficace. Des algorithmes spécialement conçus pour l'optimisation dynamique ont alors été proposés dans la littérature.

Dans cet article, nous décrivons en section 2 les principales familles de métaheuristiques utilisées en optimisation dynamique. Ensuite, notre métaheuristique MADO (*MultiAgent Dynamic Optimization*) est présentée en section 3. En section 4, nous effectuons une comparaison des meilleurs algorithmes proposés en optimisation dynamique, sur deux des principaux jeux de tests de la littérature. Enfin, nous concluons l'article en section 5.

2 Les métaheuristiques en optimisation dynamique

La plupart des algorithmes d'optimisation dynamique proposés dans la littérature sont des métaheuristiques, généralement bio-inspirées, utilisant une ou plusieurs populations de solutions. La plupart d'entre eux appartiennent à la classe des algorithmes évolutionnaires (s'inspirant de la théorie de l'évolution de Darwin) ou à celle de l'optimisation par essaim particulaire. Néanmoins, d'autres algorithmes, tels que l'optimisation par colonies de fourmis, les systèmes immunitaires artificiels et les approches hybrides, ont également été proposés. Pour une description détaillée de ces classes de métaheuristiques, on peut se référer à [1].

Il s'agit le plus souvent de métaheuristiques classiques d'optimisation statique ayant été adaptées au cas dynamique, au moyen de différentes techniques. En effet, les algorithmes d'optimisation statique ne peuvent pas en général être utilisés tels quels sur des problèmes dynamiques, principalement pour deux raisons :

1. Après un changement dans la fonction objectif, les informations que l'algorithme a pu accumuler sur celle-ci peuvent être "périmées". C'est le cas notamment des valeurs de la fonction objectif des différentes solutions archivées en mémoire. Il peut alors être nécessaire de mettre en place des traitements spécifiques, afin que le processus d'optimisation n'en soit pas perturbé. Il est possible, par exemple, de réévaluer tout ou partie des solutions mémorisées par l'algorithme.
2. Après un changement dans la fonction objectif, il est probable que l'optimum global ait changé de position dans l'espace de recherche. Il est alors nécessaire de converger rapidement vers sa nouvelle position, avant que la fonction ne change, afin de pouvoir le suivre au fil des changements de la fonction. Si la nouvelle position de l'optimum est très éloignée de la précédente, l'algorithme doit assurer une diversification suffisante pour pouvoir la trouver. Or, lorsqu'un algorithme d'optimisation statique converge vers un optimum, il peut lui être difficile de s'en écarter rapidement, afin d'explorer des zones plus prometteuses de l'espace de recherche.

Les principales techniques proposées pour permettre le suivi et la détection d'optimums peuvent être regroupées dans cinq classes [2] :

1. **Générer de la diversité après un changement :** lorsqu'un changement dans la fonction objectif est détecté, des traitements sont effectués pour augmenter la diversité des solutions et faciliter ainsi la recherche du nouvel optimum.
2. **Maintenir la diversité tout au long de la recherche :** la diversité des solutions est préservée au cours du temps, en partant du principe qu'une population largement répartie dans l'espace de recherche peut s'adapter plus efficacement aux changements.
3. **Utiliser une mémoire :** l'algorithme dispose d'une mémoire pour sauvegarder des informations sur le passé de la fonction objectif. En pratique, les bonnes solutions trouvées sont stockées, en vue d'être réutilisées lorsqu'un changement est détecté. Des techniques plus sophistiquées ont également été proposées dans la littérature [3].
4. **Utiliser plusieurs populations :** diviser la population en plusieurs sous-populations, distribuées sur différents optimums locaux, permet de suivre plusieurs optimums à la fois

et d'augmenter la probabilité d'en trouver de nouveaux.

5. **Prédire les futurs changements :** récemment, une attention particulière s'est portée sur des techniques visant à prédire les changements. Cette approche repose sur le fait que, pour des problèmes réels, les changements dans la fonction objectif sont susceptibles de suivre un certain schéma qui peut être appris.

La plupart des algorithmes d'optimisation dynamique doivent donc détecter le changement qui survient dans la fonction objectif. Le moyen le plus simple et le plus répandu pour y parvenir consiste à réévaluer une solution : si la valeur de cette solution a changé, on considère alors qu'un changement a eu lieu dans la fonction objectif. Des techniques de détection de changement plus sophistiquées ont également été proposées [4]. Néanmoins, la détection de changement par réévaluation d'une ou plusieurs solutions reste souvent la méthode la plus efficace.

3 La métaheuristique d'optimisation dynamique MADO

La métaheuristique MADO [5, 6] a été proposée pour permettre la résolution de problèmes d'optimisation continue dynamique. Il s'agit d'un algorithme multi-agents : il utilise une population d'agents pour explorer l'espace de recherche. Les agents effectuent des recherches locales en permanence : ils se déplacent pas à pas, depuis leur solution courante vers une meilleure solution avoisinante, jusqu'à ce qu'ils ne puissent plus améliorer leur solution courante, atteignant ainsi un état caractéristique d'un optimum local. Les meilleurs optimums locaux ainsi trouvés sont alors stockés en mémoire, afin d'accélérer la convergence de l'algorithme.

Les agents sont coordonnés pour explorer au mieux l'espace de recherche et garantir la diversité des solutions. La collaboration directe entre les agents se fait par l'attribution à chaque agent d'une zone de recherche locale exclusive. Ainsi, lorsque plusieurs agents convergent vers une même zone de l'espace de recherche, seul celui ayant la meilleure solution courante continue sa recherche à cet endroit. Les autres agents en conflit dans cette zone doivent redémarrer leur recherche locale ailleurs dans l'espace de recherche. C'est le cas également d'un agent ayant terminé sa recherche locale (stagnation de la solution courante). Le redémarrage de la recherche locale d'un agent se fait alors dans une zone inédite de l'espace de recherche. Lorsqu'aucune zone inédite ou dépourvue d'agent ne peut être atteinte, l'espace de recherche est considéré comme saturé d'agents. Dans ce cas, l'agent devant redémarrer sa recherche locale est détruit.

Dans MADO, la détection d'un changement dans la fonction objectif se fait par réévaluation d'une solution, sélectionnée aléatoirement parmi les solutions courantes des agents et les optimums de l'archive. Lorsqu'un changement est détecté, les solutions courantes des agents et les optimums de l'archive sont alors tous réévalués. Puis des agents sont créés et positionnés sur les optimums de l'archive. Une fois ces nouveaux agents créés, l'archive d'optimums est vidée. Cela permet le suivi, au fil des changements dans la fonction objectif, des optimums locaux trouvés.

Le schéma général de MADO est composé des deux modules suivants (voir Figure 3) :

1. **Le module de gestion de la mémoire :** il maintient l'archive contenant les optimums locaux trouvés par les agents.
2. **Le coordinateur :** il est en charge de la coordination, de la création, de la destruction et du repositionnement des agents.

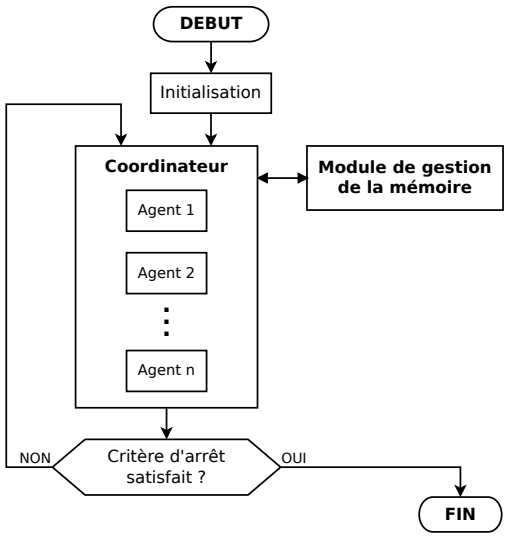

Figure 3 – Schéma général de MADO.

4 Analyse expérimentale

De nombreux problèmes réels admettent des fonctions objectifs gourmandes en temps de calcul. De ce fait, l'effort de calcul fourni par un algorithme

sur un jeu de tests est souvent exprimé en termes de nombre d'évaluations. C'est le cas pour la plupart des jeux de tests utilisés en optimisation dynamique dans la littérature.

Deux d'entre eux sont le plus souvent utilisés : le *Moving Peaks Benchmark* (MPB) et le *Generalized Dynamic Benchmark Generator* (GDBG). Nous décrivons maintenant MPB et GDBG, puis nous présentons une comparaison des algorithmes les plus compétitifs sur ces jeux de tests.

4.1 Le *Moving Peaks Benchmark*

MPB [7] est le jeu de tests le plus répandu en optimisation dynamique. Il s'agit d'un problème de maximisation, où la fonction objectif est composée d'un ensemble de pics (optimums locaux) dont la forme, la position et la taille peuvent varier au cours du temps. Lors d'un changement, n'importe lequel de ces optimums locaux peut devenir le nouvel optimum global. De plus, les déplacements aléatoires des pics sont corrélés par un coefficient λ, $0 \leq \lambda \leq 1$, où 0 correspond à des mouvements décorrélés et 1 correspond à des mouvements très corrélés. Les changements dans la fonction objectif ont lieu toutes les α évaluations.

Afin d'évaluer les performances de l'algorithme testé, une mesure appelée *offline error* est utilisée. Il s'agit de la moyenne des écarts entre la valeur de l'optimum global et celle de la meilleure solution trouvée à chaque itération. Une erreur de 0 signifie donc un suivi parfait de l'optimum.

Trois jeux de paramètres, appelés scénarios, ont été proposés pour ce jeu de tests. Le plus répandu étant le scénario 2 (voir tableau 1), c'est celui-ci que nous utilisons pour tester MADO.

Paramètre	Valeur
Nombre de pics N_p	10
Dimension d	5
Sévérité temporelle des changements α	5000
Coefficient d'autocorrélation λ	0
Nombre de changements N_c	100

Tableau 1 – Scénario 2 de MPB.

4.2 Le *Generalized Dynamic Benchmark Generator*

GDBG [8] a été utilisé lors de la compétition du congrès IEEE CEC'2009, afin de tester des métaheuristiques dédiées à l'optimisation dynamique. Ce jeu de tests est constitué de 49 fonctions. L'ensemble de ces fonctions est supposé être représentatif de la plupart des problèmes d'optimisation dynamique réels. Divers types de changements, plus ou moins chaotiques et brutaux, ont lieu après un certain nombre d'évaluations de la fonction. Ce problème GDBG est paramétrable, et comporte certains paramètres similaires à ceux de MPB. Nous utilisons ici le paramétrage de la compétition de CEC'2009 (voir tableau 2). Ce jeu de tests donne globalement un score unique sur 100 à la fin de son exécution.

Paramètre	Valeur
Dimension (fixe) d	10
Dimension (variable) d	$[5, 15]$
Sévérité temporelle des changements	$10000 \times d$
Nombre de changements	60

Tableau 2 – Paramétrage de GDBG lors de la compétition de CEC'2009.

4.3 Comparaison des performances via MPB et GDBG

Seize algorithmes ont été testés via MPB, en moyennant leurs résultats sur 50 exécutions. Les résultats des quatre algorithmes les mieux classés sont présentés dans le tableau 3. Ceux des algorithmes de la littérature testés sur GDBG sont donnés dans le tableau 4.

Ce sont des algorithmes à base de recherches locales qui obtiennent les deux premières places sur chaque jeu de tests. Parmi les algorithmes évolutionnaires testés, les mieux classés sont des algorithmes à évolution différentielle [1] : Mendes et Mohais [9] sur MPB (avec une *offline error* de 1,75 ± 0,03) et Brest *et al.* [10] sur GDBG. Parmi ceux de la classe PSO, les mieux classés sont Novoa *et al.* [11] sur MPB et Li et Yang [12] sur GDBG.

On constate que MADO est classé parmi les meilleurs algorithmes sur MPB, et qu'il obtient, sur GDBG, la deuxième place du classement, avec un score supérieur à ceux des algorithmes testés durant la compétition de CEC'2009. Ces résultats montrent l'efficacité des stratégies utilisées dans MADO.

Nous avons proposé récemment un algorithme à base de recherches locales utilisant de nouvelles stratégies [13]. Cet algorithme obtient le meilleur score sur GDBG, et la deuxième place au classement relatif à MPB.

Algorithme	*Offline error*
Moser and Chiong, 2010 [14]	0,25 ± 0,08
Lepagnot *et al.*, 2010 [13]	0,35 ± 0,06
Novoa *et al.*, 2009 [11]	0,40 ± 0,04
MADO [5, 6]	0,59 ± 0,10

Tableau 3 – Performances des quatre algorithmes les mieux classés sur MPB (classés du plus performant au moins performant).

Algorithme	Score
Lepagnot *et al.*, 2010 [13]	81,28
MADO [5, 6]	70,76
Brest *et al.*, 2009 [10]	69,73
Korosec and Silc, 2009 [15]	65,21
Yu and Suganthan, 2009 [16]	58,09
Li and Yang, 2009 [12]	57,57
França and Zuben, 2009 [17]	38,29

Tableau 4 – Performances des algorithmes testés sur GDBG (classés du plus performant au moins performant).

5 Conclusion

Dans cet article, nous présentons un état de l'art de l'optimisation dynamique, ainsi que notre métaheuristique MADO. Une comparaison des algorithmes d'optimisation dynamique proposés dans la littérature, sur les deux principaux jeux de tests, est également exposée. On remarque que les algorithmes les plus performants sur ces jeux de tests sont à base de recherches locales, et que MADO en fait partie.

Références

[1] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard, *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

[2] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments – a survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, 2005.

[3] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 12, no. 5, pp. 542–562, 2008.

[4] H. Richter, "Detecting change in dynamic fitness landscapes," in *Proc. Congr. Evol. Comput.* Trondheim, Norway : IEEE, 2009, pp. 1613–1620.

[5] J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry, "A new multiagent algorithm for dynamic continuous optimization," *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 1, pp. 16–38, 2010.

[6] J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry, "Performance analysis of MADO dynamic optimization algorithm," in *Proc. Int. Conf. Intelligent Systems Design and Applications*. Pisa, Italy : IEEE, 2009, pp. 37–42.

[7] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proc. Congr. Evol. Comput.* Washington D.C., USA : IEEE, 1999, pp. 1875–1882.

[8] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, and P. N. Suganthan, "Benchmark generator for CEC 2009 competition on dynamic optimization," University of Leicester, University of Birmingham, Nanyang Technological University, Tech. Rep., 2008.

[9] R. Mendes and A. Mohais, "DynDE : A differential evolution for dynamic optimization problems," in *Proc. Congr. Evol. Comput.* Edinburgh, Scotland : IEEE, 2005, pp. 2808–2815.

[10] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, and V. Zumer, "Dynamic optimization using self-adaptive differential evolution," in *Proc. Congr. Evol. Comput.* Trondheim, Norway : IEEE, 2009, pp. 415–422.

[11] P. Novoa, D. A. Pelta, C. Cruz, and I. G. del Amo, "Controlling particle trajectories in a multi-swarm approach for dynamic optimization problems," in *Proc. Int. Work Conf. Interplay Between Natural Artif. Comput.* Santiago de Compostela, Spain : Springer, 2009, pp. 285–294.

[12] C. Li and S. Yang, "A clustering particle swarm optimizer for dynamic optimization," in *Proc. Congr. Evol. Comput.* Trondheim, Norway : IEEE, 2009, pp. 439–446.

[13] J. Lepagnot, A. Nakib, H. Oulhadj, and P. Siarry, "A multiple local search algorithm for continuous dynamic optimization," under submission.

[14] I. Moser and R. Chiong, "Dynamic function optimisation with hybridised extremal dynamics," *Memetic Computing*, vol. 2, no. 2, pp. 137–148, 2010.

[15] P. Korosec and J. Silc, "The differential ant-stigmergy algorithm applied to dynamic optimization problems," in *Proc. Congr. Evol. Comput.* Trondheim, Norway : IEEE, 2009, pp. 407–414.

[16] E. L. Yu and P. Suganthan, "Evolutionary programming with ensemble of external memories for dynamic optimization," in *Proc. Congr. Evol. Comput.* Trondheim, Norway : IEEE, 2009, pp. 431–438.

[17] F. O. de França and F. J. V. Zuben, "A dynamic artificial immune algorithm applied to challenging benchmarking problems," in *Proc. Congr. Evol. Comput.* Trondheim, Norway : IEEE, 2009, pp. 423–430.