Évolution d'algorithmes de recherche locale.

Vince Hénaux, Adrien Goëffon, Frédéric Saubion

Université d'Angers, LERIA
{prenom.nom}@univ-angers.fr

Mots-clés : *évolution artificielle, recherche locale, métaheuristiques, optimisation discrète*

1 Introduction

La résolution approchée d'un problème d'optimisation combinatoire par un algorithme de recherche — qu'il s'agisse d'un algorithme génétique, d'un algorithme de recherche locale ou d'un algorithme basé sur un modèle de substitution — consiste à échantillonner l'espace de recherche au moyen d'une stratégie d'apprentissage, déterminant les prochaines solutions à évaluer en fonction des solutions précédemment évaluées. Classiquement, on évalue la performance d'un algorithme selon sa capacité à trouver de bonnes solutions sur un ensemble d'instances partageant certaines similitudes. Un effort particulier est dédié aujourd'hui à concevoir et utiliser des techniques de configuration ou de paramétrage automatique d'algorithmes, permettant à un algorithme d'obtenir des performances optimales sur un sous-ensemble d'instances. Dans un tel contexte de configuration automatique d'algorithmes [2], appliquer une telle stratégie pour construire un algorithme de résolution optimal pour une instance donnée est assimilé à un surapprentissage [1]. Cela revient, étant donné un modèle d'algorithme défini en amont, à déterminer un algorithme ayant une performance optimale pour résoudre une instance donnée [3]. Or trouver un tel algorithme revient à résoudre l'instance initiale, sauf que l'espace de recherche dans lequel on se place (l'espace des algorithmes plutôt que l'espace des solutions) est alors différent.

Les algorithmes de recherche de type métaheuristiques et recherche locale se basent sur une évolution de la solution (ou de la population) courante au moyen d'un guidage par une fonction d'évaluation. Usuellement, cette fonction d'évaluation est ou découle directement de la fonction objectif du problème. Les difficultés de résolution apparaissent alors lorsque les couples solution / valeur objectif décrits par l'instance perdent en interprétation et semblent véhiculer des informations contradictoires, c'est-à-dire lorsque paysage de recherche naturellement induit par l'instance de problème n'est pas parfaitement exploitable, présente un certain niveau de rugosité et donc comporte de nombreux optimums locaux. Nous proposons ici d'étudier des moyens de construire une fonction d'évaluation permettant une résolution plus aisée d'une instance de problème par un algorithme de recherche locale simple. C'est ainsi que l'on déplace la problématique de la recherche d'une solution, à celle de la recherche d'un algorithme dont le modèle serait donné (par exemple, un simple hill-climbing ou *climber*) et dont le guidage serait fonction d'une fonction d'évaluation à déterminer.

2 Évolution artificielle d'un climber

Un algorithme d'échantillonnage de l'espace des solutions par recherche locale peut se définir au moyen de quatre éléments : un espace de recherche, une relation de voisinage, une fonction d'évaluation et une stratégie de guidage. Trouver un bon algorithme consiste bien souvent à trouver une bonne stratégie de guidage, permettant notamment de perturber la recherche autour des optimums locaux, nécessaire lorsque le paysage de recherche présente un certain niveau de rugosité. Ici, outre l'espace de recherche (correspondant à celui de l'instance à résoudre) et la relation de voisinage, nous fixons la stratégie de guidage au simple mécanisme

de type hill-climbing. L'unique paramètre de l'algorithme est donc la fonction d'évaluation sur laquelle porte le guidage. Le problème revient dès lors à déterminer, dans un espace de climbers dont seule la fonction d'évaluation n'est pas connue, un algorithme efficace pour optimiser une fonction objectif. On cherche alors implicitement à construire le paysage de recherche tel qu'un climber aura tendance à atteindre, au final, les meilleures solutions de l'instance à résoudre. L'idée sous-jacente est bien de reformuler, dans un autre modèle de représentation, la problématique de la recherche d'une solution du problème initial, ici sans aborder explicitement certaines difficultés inhérentes à la fonction objectif, et en particulier son adéquation avec l'espace de recherche et la relation de voisinage.

Faire évoluer un climber consiste donc ici à faire évoluer une fonction d'évaluation selon une stratégie d'évolution classique. Étant donné un modèle de fonction, l'algorithme commence par générer une fonction aléatoire puis, à chaque itération de l'algorithme, mute la fonction courante et valide la mutation si le climber guidé par cette fonction mutée est plus performant que le climber guidé par la fonction courante. L'algorithme stoppe après un certain nombre de mutations non améliorantes. Le climber ainsi obtenu est l'algorithme de recherche basé sur la fonction d'évaluation évoluée. On mesure la performance de cet algorithme de recherche en estimant l'espérance de la valeur objectif (au sens de la fonction objectif) d'un optimum local (au sens de la fonction d'évaluation).

3 Expérimentations

Dans ces travaux préliminaires, nous restreignons l'étude aux fonctions NK, dont les propriétés structurelles peuvent être étudiées par ailleurs (rugosité des paysages induit, distribution des optimums locaux, *etc.*). Ainsi les instances de problèmes étudiées sont des fonctions NK variées, et les modèles de fonction d'évaluation sont des fonctions NK particulières ($K = 1$, c'est-à-dire des fonctions NK compressées au maximum et ayant la particularité de définir des paysages moins rugueux).

La table 1 compare les performances obtenues par les climbers générés, donc guidés par des fonctions d'évaluation alternatives déterminées par un mécanisme évolutionnaire, à celles obtenues par les climbers de référence, guidés par les fonctions objectif. On remarque que le processus d'apprentissage ainsi défini permet de générer dans tous les cas des climbers plus performants que les climbers de référence, en modifiant uniquement la fonction d'évaluation guidant la recherche.

Instance	HC(f_{obj})	HC(f_{alt})
nk-128-1	0.709	**0.718**
nk-128-2	0.716	**0.726**
nk-128-4	0.727	**0.732**
nk-128-6	0.717	**0.728**
nk-128-8	0.715	**0.721**
nk-128-10	0.708	**0.714**
nk-128-12	0.700	**0.708**

Instance	HC(f_{obj})	HC(f_{alt})
nk-256-1	0.691	**0.703**
nk-256-2	0.713	**0.723**
nk-256-4	0.719	**0.731**
nk-256-6	0.723	**0.733**
nk-256-8	0.717	**0.722**
nk-256-10	0.710	**0.720**
nk-256-12	0.707	**0.712**

TAB. 1 – Performance d'un climber guidé par une fonction d'évaluation alternative, par rapport à un climber de référence.

Références

[1] Sumio Fujita. Retrieval parameter optimization using genetic algorithms. *Information Processing & Management*, 45(6) :664–682, Elsevier, 2009.

[2] Holger H. Hoos. Automated algorithm configuration and parameter tuning. *Autonomous search*, 37–71, Springer, 2011.

[3] John R. Rice. The Algorithm Selection Problem. *Advances in computers*, 1565–118, Elsevier, 1976.