Local Search Solver, une bibliothèque de recherche locale pour des applications industrielles

Florian Fontan[1], Luc Libralesso[2]

[1] dev@florian-fontan.fr

[2] University Clermont Auvergne, LIMOS, CNRS UMR 6158, Aubière France

Mots-clés : *recherche locale, challenge ROADEF/EURO 2020.*

La recherche locale fait partie des principales méthodes de résolution de problèmes d'optimisation combinatoire. Elle est par exemple utilisée pour résoudre des problèmes de tournées de véhicules [5, 4], des problèmes d'ordonancement [2], des problèmes de packing [6], des problèmes dans des graphes [1, 3, 7], etc.

Toutefois, résoudre un problème avec cette technique demande en général un temps de développement supérieur au temps nécessaire à l'écriture d'un modèle en Programmation Linéaire en Nombres Entiers ou d'un modèle en Programmation Par Contrainte. Ceci rend son utilisation difficile en pratique pour des problèmes complexes dont le modèle évolue rapidement.

Nous cherchons donc à développer des méthodes et outils basés sur la recherche locale présentant un bon compromis entre le temps nécessaire au développement et la qualité des solutions renvoyées.

Nous présentons une bibliothèque écrite en C++ [1] publiée sous licence MIT, dont l'objectif est de permettre l'implémentation rapide d'algorithmes performants basés sur la recherche locale. L'utilisateur n'a besoin d'écrire que les composants spécifiques au problème considéré. La bibliothèque contient des implémentation génériques des algorithmes classiques de recherche locale avec redémarrages, recherche locale itérée, et recherche locale génétique, que l'utilisateur n'a pas besoin d'implémenter à nouveau.

La bibliothèque contient également un algorithme dérivé de la recherche locale itérée que nous appelons "Best First Local Search", présenté dans l'Algorithme 1. Cette variante combine le concept de perturbations de la recherche locale itérée avec les méthodes de recherche arborescente. Au lieu d'avoir une unique solution courante, toutes les solutions sont stockées dans une file, et, à chaque itération la solution la plus prometteuse est extraite et ses solutions filles sont générées en appliquant des perturbations et ajoutées à la file. L'objectif est d'avoir un algorithme qui fonctionne au mieux avec peu de voisinages, et des voisinages simples à implémenter.

Algorithm 1 Best First Local Search

```
function BestFirstLocalSearch(instance)
    queue ← empty queue of solutions
    Add an initial solution to the queue
    while queue is not empty do
        solution ← extract "most promising" solution from queue
        Run local search on solution
        for each perturbation do
            child ← apply perturbation to solution
            Add child to queue
```

Nous avons notamment testé cette approche lors du challenge ROADEF/EURO 2020 portant sur un problème industriel complexe de planification de maintenances.

1. https://github.com/fontanf/localsearchsolver

Dans notre implémentation, le seul voisinage utilisé est le voisinage "shift" qui consiste à déplacer une intervention. Le voisinage est exploré avec une stratégie "best improvement". La perturbation consiste également en un shift : la date de début d'une intervention est forcée. L'implémentation spécifique au problème demande de l'ordre de 1000 lignes de code en C++. Elle a été soumise lors de la phase finale de la compétition.

Bien que cette approche n'aie pas été la meilleure, elle a quand même permis de trouver 3 meilleures solutions sur 30, et se classe 4^{e} sur 13 lors de la phase finale.

Cette bibliothèque inclut également un module dédié aux problèmes de séquences. Dans ce cas, l'utilisateur fournit principalement une fonction qui ajoute un élément à une sous-séquence. L'ensemble des voisinages, perturbations et algorithmes de croisement classiques pour ce genre de problèmes sont implémentés : shifts, swaps, 2-opt...À la place de cette, fonction, il est également possible de fournir une fonction qui concatène deux sous-séquences, afin de profiter des évaluations en temps constant comme décrit par [5]. Ce module permet d'avoir un algorithme performant pour ces problèmes en écrivant seulement une centaine de lignes de code.

Références

[1] Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered : The Winning Solver from the PACE 2019 Challenge, Vertex Cover Track. In *2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing (CSC)*, Proceedings, pages 1–11. Society for Industrial and Applied Mathematics, January 2020.

[2] Arthur Kramer and Anand Subramanian. A unified heuristic and an annotated bibliography for a large class of earliness–tardiness scheduling problems. *Journal of Scheduling*, 22(1) :21–57, February 2019.

[3] Xiangjing Lai, Jin-Kao Hao, Zhang-Hua Fu, and Dong Yue. Neighborhood decomposition-driven variable neighborhood search for capacitated clustering. *Computers & Operations Research*, 134 :105362, October 2021.

[4] Puca Huachi Vaz Penna, Anand Subramanian, Luiz Satoru Ochi, Thibaut Vidal, and Christian Prins. A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. *Annals of Operations Research*, 273(1) :5–74, February 2019.

[5] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3) :658–673, May 2014.

[6] Zequn Wei and Jin-Kao Hao. A threshold search based memetic algorithm for the disjunctively constrained knapsack problem. *Computers & Operations Research*, page 105447, July 2021.

[7] Xinyun Wu, Zhipeng Lü, and Fred Glover. A Fast Vertex Weighting-Based Local Search for Finding Minimum Connected Dominating Sets. *INFORMS Journal on Computing*, November 2021. Publisher : INFORMS.