POLYTECHNIQUE
MONTRÉAL

UNIVERSITÉ
D'INGÉNIERIE

École Polytechnique de Montréal

INF6102 - Métaheur. applic. au génie informatique

Hiver 2025

Projet
Eternity II

Groupe : 01

2311468 - Jean BARRETO

16 Avril 2025

Table des matières

1 Architecture Globale

L'idée de l'algorithme est de partir d'une solution Greedy puis d'essayer de l'améliorer via une succession de Destruction/Reconstruction/Tabu Search en augmentant progressivement le ratio de Destruction jusqu'à un seuil, puis de repartir d'une toute nouvelle solution Greedy et de recommencer. On garde ainsi la meilleur solution obtenu depuis le début.

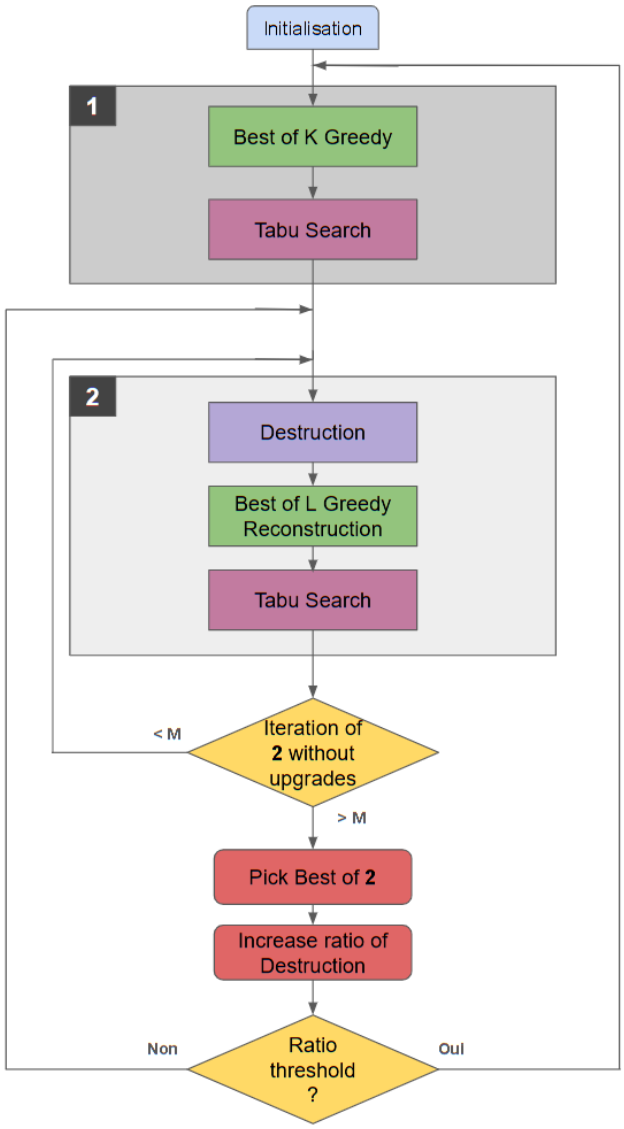

FIGURE 1 – Architecture Globale - On sort dès que le temps limite est atteint

2 Blocs principaux

2.1 Greedy Search

Le Greedy Search Search commence toujours par mélanger les pièces non utilisées (afin de générer des solutions différentes) puis il parcours le puzzle diagonales par diagonales en partant du coin en bas à gauche. Lorsqu'il tombe sur une case vide il y place la pièces non utilisé qui engendre le moins de conflits à cette endroit.

Son intérêt principale est sa rapidité accrue, ce qui permet de générer des milliers de solutions possible en quelques secondes. Le tableau ci dessous donne le temps d'exécution de 1000 Greedy Search sur un puzzle vide selon l'instance.

	Instance A	Instance B	Instance C	Instance D	Instance E
Temps d'éxecution (s)	0.24	1.59	4.06	7.39	11.75

TABLE 1 – Vitesse d'exécution de 1000 Greedy à partir d'un puzzle vide

Remarquons que dans l'algorithem principale, les Greedy Search se ofnt sur des puzzles en grande majorité déjà rempli ce qui accélère grandement le temps d'éxécution (facteur d'environ 10 selon les ittérations).

2.2 Tabu Search

La recherche Tabu part d'un puzzle complet et tente de l'améliorer petite à petit.

Espace de recherche : L'espace de recherche est composé de toute les dispositions de pièces tels que les pièces au bords n'ont aucun conflit avec de dernier.

Voisinage : On considère 2 types d'opérations de voisinage. La première, "Rot", est une simple rotation d'une pièce. La seconde, "2-Swap", est un échange de 2 pièces homogène (ayant le même nombre de coins gris) avec rotations de chacune des pièces. Ainsi si n est la dimension du puzzle, la première opération donne $3.(n-2)^2$ (les pièces du bords n'ont qu'une orientation cohérente avec ce dernier donc on ne les tourne pas). La seconde opération engendre $4^2.(\binom{4}{2} + \binom{4.(n-2)}{2} + \binom{(n-2)^2}{2})$ voisins, cela correspond au nombre de façon de choisir 2 coins, 2 bords ou 2 pièces intérieurs multiplier par 4^2 le nombre de combinaison de rotations entre ces 2 pièces. Ce voisinage est connexe

dans le sens où l'on peut de manière assez intuitive explorer tout l'espace de recherche en passant de voisin en voisin.

Evaluation : Une solution voisine est évaluée par le nombre total de conflits qu'elle possède

Sélection : On séléctionne le voisin ayant la plus petite évaluation

Abstraction d'une solution : Lors d'un mouvement "2-Swap" d'une solution à une autre, la représentation que l'on ajoute à notre *tabu_list* est un tuple $(x1, y1, p1, o1, x2, y2, p2, o2)$ où (p1, o1) est la pièce et l'orientation de la pièce qui se retrouve en (x1, y1) et (p2, o2) la pièce et l'orientation de la pièce qui se retrouve en (x2, y2). Cette abstraction est relativement forte car elle ne considère que 2 pièces du puzzle.

Critère d'Aspiration : Un mécanisme d'aspiration permettant d'accepter un voisins même si il est tabu dans le cas où il est meilleure que notre meilleur solution actuelle.

Hyperparamètres : On peut désigner 3 hyperparamètres à cette recherche tabu. **TABU_MAX_ITER** le nombre d'ittérations sans améliorations avant de sortir de la recherche tabu, **TABU_LENGTH** la longueur de la list tabu (et donc le nombre d'ittération où une solution est bannie) et enfin **TABU_RATIO** la proportion du voisinages considéré.

2.3 Destruction

La fonction destruction pars d'un puzzle complet et supprime au hasard un pourcentage des pièces qui n'ont aucun conflits (**UNCONFLICT_RATIO**) et un autre pourcentage (**CONFLICT_RATIO**) de pièces qui ont au moins un conflits pondéré par leurs nombres de conflits. Ainsi une pièces à 4 conflits aura plus de chance d'être retirée qu'une à 1 conflit.

CONFLICT_RATIO à typiquement une valeur de 0.8 afin de retirer la majorité des pièces en conflits pour s'extraire d'un minimum local alors que **UNCONFLICT_RATIO** à une valeur prgressive entre 0.1 et **RATIO_TRESHOLD**, au delà on considère que l'on retire trop de pièce sans conflits pour garder une cohérence sur la solution de départ et qu'il est équivalent de vider le puzzle en entier et de repartir sur une Greedy Search.

3 Hyperparamètres

Avant de commencer cette section il est important de noter plusieurs remarque du calibrage des hyperparamètres :

— **Première remarque :** Soit H_set_X l'ensemble des valeurs des hyperparamètres ofrant les meilleures performances l'instance X. A priori rien ne garanti que H_set_X est identique pour toute les insatnces X. Ainsi il faudrait optimiser H_set_X pour chacune des instances X.

— **Deuxième remarque :** L'espace de recherche des meilleures hyperparamètres pour une instance est énorme. Soit H_i l'hyperparamètres numéro i, bien souvent H_i peut prendre des valeurs continue dans un intervalle plus ou moins grand (voir infini).

— **Troisième remarque :** L'espace de recherche des hyperparamètres n'est à priori pas convexe, ce qui rend une recherche rapide impossible.

Au vu de cela, la suite de la section se permet de prendre de nombreuses hypothèses simplificatrice et ne se concentre que sur l'optimisation de quelques hyperparamètres.

3.1 Hyperparamètres Tabu search

Il y a un total de 3 hyperparamètres pour la Tabu Search : $TABU\_MAX\_ITER\_WITHOUT\_UPGRADE$, $TABU\_LENGTH$ et $TABU\_RATIO$ (voir section consacré pour la définition). Fixons $TABU\_MAX\_ITER\_$ à 1000 et $TABU\_RATIO$ à 0.2 afin d'avoir des ittérations rapide et de finir l'éxecution de la tabu rapidement. Les résultats suivant ont été obtenu sur l'instance B, avec une graine aléatoire fixe, sur 100 Tabu Search en faisant varier l'hyperparamètre $TABU\_LENGTH$.

	10	50	200	500
Conflits moyen	12.13	9.23	9.30	9.93

TABLE 2 – Conflits moyen sur 100 Tabu Search selon $TABU\_LENGTH$ avec $TABU\_RATIO = 0.2$ et $TABU\_MAX\_ITER\_WITHOUT\_UPGRADE = 1000$

On remarque bien un minimum de conflits pour une taille de liste tabu autour de 50. C'est donc la valeur que l'on gardera par la suite. Malgrès la remarque 3 en début de section nous allons faire cette fois ci faire varier $TABU\_RATIO$ dans le mêmes conditions.

	0.1	0.2	0.4
Conflits moyen	10.41	9.23	9.58

TABLE 3 – Conflits moyen sur 100 Tabu Search selon *TABU_RATIO* avec *TABU_LENGTH* = 50 et *TABU_MAX_ITER_WITHOUT_UPGRADE* = 1000

Cette fois ci on trouve un minimum autour de 0.2 c'est donc la valeur que l'on gardera.

3.2 Hyperparamètres Globales

Fixons **K**, **L**, **M** (voir schéma architecture globale) respectivement à 1000, 100 et 10 ainsi que **CONFLICT_RATIO** à 0.8. On s'interresse au meilleurs score obtenu en moyenne pour différent **UNCONFLICT_RATIO** allant de 0.1 à 0.8.

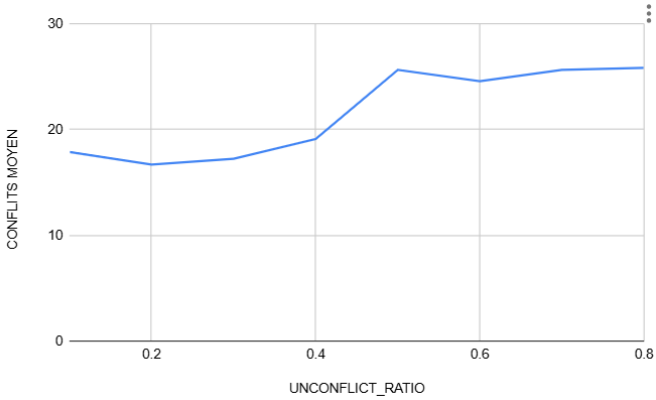

FIGURE 2 – Evolution du nombre de conflits moyen selon **UNCONFLICT_RATIO**

On voit un net palier à partir de 0.4, cela s'explique par le fait que si on retire trop ce pièces qui n'étaient pas en conflits, on perds trop de cohérence sur notre solution initiale. On pose alors **RATIO_TRESHOLD** à 0.4.

4 Solver Heuristic & Solveur Local Search

4.1 Solver Heuristic

Ce solveur parcours les couronnes du puzzle depuis la couronne extérieur (le bord) à intérieur (le centre). Pour chaque couronne il fait un placement Greedy avec les pièces qu'il lui reste à disposition.

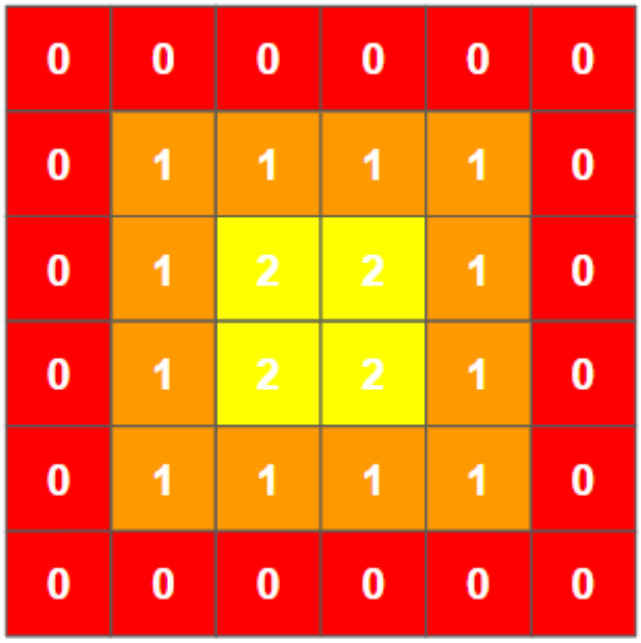

FIGURE 3 – Ordre de parcours des couronnes

Pour chaque couronne on parcours les cases de gauche à droite puis de haut en bas en partant d'en bas à gauche.

4.2 Solveur Local Search

Ce solveur effectur une recherche local sur le principe du Simulated Annealing. Si le meilleur voisin améliore la solution alors il le prend automatiquement, sinon il le prend avec une proba $\exp\left(-\frac{\Delta}{T}\right)$ où Δ est la différence de conflit entre le meilleur voisin et la solution actuelle et T la température qui décroit selon un schéma géométrique $T_{i+1} = \alpha T_i$ avec $T_0 = 100$ et $\alpha = 0.99$

Le voisinage est le même que pour la Tabu Search de l'implémentation Advanced

5 Résultats et discussions

Voici le récapitulatif des meilleures scores obtenues sur les instances principales pour les différentes implémentations :

	Instance A	Instance B	Instance C	Instance D	Instance E
Heuristic	0	24	23	42	40
Local Search	0	20	30	50	61
Advanced	0	0	8	10	10

TABLE 4 – Meilleurs scores

Plusieurs phénomènes sont à noter. Tout d'abbord sans surprise toute les implémentations résolvent l'instance A de par sa taille très abordable. Ensuite on pourrait penser que de par sa simplicité, l'implémentation Heuristic soit pire que l'implémentation Local Search, cependant elle est en réalité meilleure (ou au pire équivalente).

C'est en se basant sur ce constat que ma première idée pour l'implémentation Advanced comportati cette logique de construction couronne par couronne, mélangé à divers métaheuristique. Cependant après plusieurs jours j'ai constaté que malgrès l'aspect prometteur de cette conception par couronne je n'arrivait pas à la faire progresser tant que ça. Je l'ai donc mise de côté au profit de l'implémentation Advanced présenté dans ce rapport.

Choix qui semble avoir porter ses fruits au vu des score de cette dernière, Bien que je sois assez content des score de Advanced sur D et E je reste assez décu de ne pas avoir résolu le C. Je note également que je n'ai réussi à résoudre la B que 2 fois sur mes nombreux essais, le reste du temps Advanced retournait une solution entre 2 et 4 conflits.

Je tiens également à resouligner la difficulté à trouver de bon hyperparamètres, je pense qu'une implémentation plus simple avec moins d'hyperparamètres aurait plus de chance d'être mené à bien. J'en prend note pour l'avenir.