# Recherche de solutions au problème de coloration équitable d'un graphe

Alexandre ROBIN
Université d'Angers, Master 2 Recherche Intelligence Décisionnelle

***Résumé*—Le problème de coloration équitable d'un graphe est un problème étendu de la coloration de graphe qui peut se retrouver dans divers domaines variés tels que des problèmes de planification. Dans ce papier, nous verrons donc une solution que nous avons pu construire et comparerons nos résultats obtenus sur différents benchmarks.**

## I. Introduction

Considérons un ensemble de sommets $V$ d'un graphe $G = (V, E)$. Si cet ensemble peut se retrouver partitionné en $k$ différentes classes $C = C_1, ..., C_i, ..., C_k$ de telle sorte que pour chaque couple $(C_i, C_j)$, la condition de répartition $||C_i| - |C_j|| \leq 1$ soit respectée, alors nous aurons obtenu une k-coloration équitable. Dans un problème de coloration de graphe dit "classique" nous avons $\chi(G) = k$, le plus petit nombre de couleurs possibles pour colorer un graphe, appelé nombre chromatique. Ici, $\chi_e(G) = k$ est appelé nombre chromatique équitable et correspond au plus petit nombre de couleurs possibles tout en intégrant la condition de répartition pour colorer un graphe. Le problème de coloration équitable n'étant qu'une extension du problème de coloration grâce à l'ajout d'une contrainte additionnelle sur la répartition des couleurs, nous pouvons donc déduire que pour n'importe quel graphe $\chi(G) \leq \chi_e(G)$.

Cette méthode de coloration peut-être perçue dans beaucoup de problèmes réels et naturels. En effet, dans les problèmes de planification, une tâche peut-être représentée par un sommet et deux sommets reliés signifient que ces tâches doivent être réalisées à des moments différents pour un manque de ressources ou de personnels par exemple. Nous chercherons donc à avoir une coloration possible de telle sorte que les tâches adjacentes ne soient pas réalisées en même temps ainsi qu'une répartition équitable ou quasi-équitable à $\pm 1$ une tâche près pour que tous les employés soient sur un pied d'égalité. Un autre problème qui pourrait être modélisé est celui de l'emploi du temps des professeurs et élèves d'une université par exemple.

Aujourd'hui, ce problème étant relativement connu, de nombreuses solutions sont déjà présentes dans la littérature à base d'algorithmes génétiques constitués d'opérateurs de croisement et mutation bien propres à ce problème. De même, des recherches tabous intégrant du backtracking ont déjà prouvé leur efficacité ainsi que différentes techniques basées sur des algorithmes gloutons.

Dans ce papier, nous allons tout d'abord voir notre recherche de solutions à ce problème, avec la modélisation du problème ainsi que les algorithmes utilisés, puis après une implémentation, nous pourrons comparer les résultats obtenus par rapport à ceux présents dans la littérature pour valider ou rejeter cette solution.

## II. Présentation de notre solution

Notre solution va reposer sur une modélisation similaire en certains points à un algorithme génétique (AG). Ceux-ci sont aujourd'hui utilisés dans de nombreux domaines de recherche depuis que leur efficacité n'est plus à prouver pour certains types de problèmes et sont définis selon différents points :

- Un individu qui représente une solution potentielle au problème donné et qui dans ce problème sera constitué d'un graphe ;
- Une population qui est composée de tous les individus présent dans l'espace de recherche ;
- Une fonction d'évaluation qui sera aussi appelée fonction de fitness et qui permet de définir une performance aux individus ;
- Des opérateurs nous permettant de faire évoluer nos individus dans la population.

### A. *Individu et Population*

Chaque problème possédera la plupart du temps sa propre structure donnée au niveau des individus. En effet, le codage doit se retrouver être en parfaite cohésion avec le problème pour permettre ainsi une utilisation "facile" de l'algorithme et ainsi trouver une solution optimale.

Notre individu sera donc ici composé d'un graphe qui sera lui-même représenté par une matrice permettant de connaître les différentes arêtes entre les sommets du graphe. De plus, nous aurons donc un tableau des différents sommets ainsi qu'un tableau des différentes couleurs utilisées dans le graphe. Pour permettre une utilisation facile de la solution, un langage orienté objet sera utilisé et un sommet sera donc défini par une valeur, une couleur, un degré ainsi qu'un degré de saturation. Le degré de saturation sera expliqué lors de la phase d'initialisation ; le degré est lui, le nombre de sommets adjacents à celui-ci. Les différentes couleurs utilisées seront représentées par des nombres entiers et chaque sommet appartiendra donc à ce que l'on appelle une classe de couleur.

Ces classes de couleurs pourront-être calculées tout au long de l'algorithme dès que l'on en définira le besoin et seront constituées des différents sommets la composant.

Une population se révèle être constituée de différents individus qui évolueront. Dans cette solution, nous verrons que ces individus seront tous indépendants les uns des autres durant tout l'algorithme. Cette population sera de taille fixe du début à la fin de l'AG. Plusieurs initialisations des individus d'une population sont possibles : une heuristique réalisée au préalable pour insérer des individus "bons" dès le départ que nous réaliserons ici, un tirage aléatoire des valeurs des individus...

### B. *Évaluation d'un individu*

L'évaluation d'un individu correspond à son efficacité face au problème et est exprimée en valeur positive entière. Elle est calculée grâce à une fonction appelée fonction de fitness. Lors d'un problème de coloration "classique", cette fonction peut-être simplement définie par le nombre de couleurs $k$ présentes dans le graphe de l'individu si un graphe possède toujours une coloration possible, une pénalité peut sinon être ajoutée si le graphe possède des conflits entre voisins. Ici, nos graphes seront toujours composés de coloration possible. De plus, il est nécessaire d'introduire un nouveau paramètre permettant de prendre en compte la condition de répartition des couleurs pour nous permettre d'atteindre une coloration équitable.

Chaque classe de couleur sera donc calculée avec les sommets la constituant. Une moyenne $m$ du nombre de sommets présents dans chaque classe de couleur est faite $m = 1/k * \sum_{i=1}^{k} |C_i|$ et un calcul de différence est réalisé entre cette moyenne et chaque classe de couleur. Ces différences sont accumulées si et seulement si elles se trouvent être supérieures à 1 de telle sorte à obtenir une pénalité représentée par un entier positif supérieur à 0 $p = \sum_{i=1}^{k} |m - |C_i||$ si $|m - |C_i|| > 1$.

Cette pénalité est ensuite simplement divisée par 3 puis ajoutée aux nombres de couleurs utilisées dans le graphe pour obtenir l'évaluation de sa performance et donc sa fitness. Plus un graphe possédera une évaluation faible et plus celui-ci sera meilleur. Nous nous situons donc dans un problème de minimisation.

### C. *Initialisation*

L'initialisation peut-être réalisée de différentes manières. Ici, nous avons choisi d'utiliser une heuristique appelée DSATUR. Nous allons tout d'abord commencer par expliquer le calcul du degré de saturation d'un sommet ainsi que la clique maximale d'un graphe.

*1) **Notion de clique**:* L'appellation clique correspond à un sous-ensemble des sommets d'un graphe dont le sous-graphe induit est complet, c'est-à-dire que deux sommets quelconques de la clique sont toujours adjacents et ne peuvent donc obtenir la même couleur.

Une information très importante pour le départ. En effet nous savons que chaque sommet nécessite une couleur différente et nous pouvons donc affirmer que $\omega(G) \leq \chi(G)$ sachant que $\chi(G) \leq \chi_e(G)$ alors $\omega(G) \leq \chi_e(G)$ pour $\omega(G)$, la taille de la plus grande clique du graphe. Nous cherchons donc un sous-graphe qui contient le plus de sommets possibles. Le calcul de la clique maximale étant lui-même un problème difficile, nous fixerons un nombre d'itérations maximales pour son calcul.

L'algorithme de calcul se déroulera comme suit : nous chercherons une première clique du graphe puis nous entrerons dans une boucle où nous retirerons deux sommets aléatoirement de cette clique pour permettre l'ajout de nouveaux sommets. Tous ces nouveaux sommets possibles pour former une nouvelle clique sont ajoutés et nous comparons cette nouvelle clique à l'ancienne. Si la nouvelle se retrouve être plus grande en nombre de sommets $Vi$ présents que l'ancienne, alors une sauvegarde de cette nouvelle clique est réalisée et le procédé continue ensuite sur la nouvelle clique calculée, et cela pendant le nombre d'itérations fixées.

Enfin, afin d'essayer d'améliorer au maximum la meilleure clique trouvée, nous bouclerons sur cette dernière et supprimerons les sommets un à un pour, de même que précédemment, espérer pouvoir en ajouter plus à chaque suppression. Lorsqu'une nouvelle meilleure clique est trouvée durant ces itérations, nous recommençons entièrement une nouvelle boucle sur cette nouvelle clique. Si cette dernière ne peut-être améliorée par la suppression un à un de ces sommets, alors nous considérerons cette meilleure clique en tant que clique "optimale" pour notre résolution.

*2) **Notion de degré de saturation**:* Le degré de saturation $DSAT(Vi)$ avec $Vi$ un sommet d'un graphe G est calculé d'une toute autre manière. Si aucun voisin de $Vi$ n'est colorié, alors son degré de saturation est égal à son degré et nous avons donc $DSAT(Vi) = degre(Vi)$. A savoir que le degré d'un sommet se trouve être le nombre d'arêtes reliées à ce sommet et donc tout naturellement le nombre de voisins que possède ce sommet.

En revanche si $Vi$ possède des voisins coloriés, alors son degré de saturation sera égal aux nombre de couleurs présentes dans ses voisins.

*3) **L'algorithme DSATUR**:* Maintenant que nos principaux paramètres de l'algorithme peuvent-être calculés, voyons la manière des les utiliser. Nous commençons tout d'abord par colorier la clique maximale trouvée avec différentes couleurs puis par trier dans une liste les sommets non colorés par ordre de degré de saturation décroissant. Ensuite nous colorons donc le premier sommet de cette liste avec la plus petite couleur possible en fonction de ses voisins. Nous devons maintenant mettre à jour le degré de saturation de tous ses sommets voisins.

En effet, la couleur ayant été déterminée, les voisins voient leur DSAT se modifier. Puis nous recommençons à sélectionner un nouveau sommet dans notre liste triée jusqu'à ce que tous les sommets soient colorés.

Cette initialisation sera réalisée sur le graphe de chaque individu pour permettre d'obtenir des colorations plus ou moins différentes entre chacun d'eux dû à un calcul de la clique maximale "approximatif" où nous y avons inséré une part d'aléatoire.

Nous avons pu voir qu'ici, seulement le problème de coloration est pris en compte et non pas une coloration équitable. Cette extension sera gérée par un opérateur que nous allons voir dans la partie suivante. En revanche, nous obtenons souvent des résultats déjà plutôt satisfaisants dans l'obtention d'un nombre chromatique $\chi(G)$ le plus faible possible pour un graphe $G$.

### D. *Opérateurs*

Après la phase d'initialisation, nous possédons maintenant des individus dotés d'une "bonne" qualité, un AG "basique" devrait donc reposer sur différentes phases qui seront répétées jusqu'au critère d'arrêt choisi :

- Sélection
- Croisement
- Mutation
- Insertion

Nous allons voir que dans notre algorithme, semblable à un algorithme génétique, nous ne posséderons pas de croisements. En effet, lors de nos différents tests, nous nous sommes aperçus que ceux-ci dégradaient trop nos individus et qu'une reconstruction par la suite était trop longue à réaliser. De ce fait, nous avons pris la décision de ne pas réaliser de croisements ici. Les individus ne partageront donc aucune information génétique entre eux et se contenteront de mutations qui permettront leur évolution, chacun de leur côté.

*1) **Sélection**:* Dans la littérature, de nombreux opérateurs de sélection existent tel que le fait de choisir les $k$ individus ayant la meilleure fitness. L'inconvénient de cette méthode est une convergence trop rapide vers un individu qui n'est pas forcément l'optimum global et cela peut, par la suite, engendrer une perte de diversité dans la population.
Une sélection complètement aléatoire pourrait, elle aussi, être choisie mais cette fois-ci nos individus évolueront trop lentement.
Une alternative possible à ce problème est la sélection dite par tournoi. L'avantage de cette méthode est que la sélection reste basée sur la performance d'un individu, mais, en revanche, celle-ci tire au hasard deux individus dans la population et les compare entre eux. Le meilleur des deux est ensuite choisi pour accéder à la phase suivante.

Cette procédure de sélection peut-être réalisée autant de fois que l'on souhaite sélectionner des individus. Pour le moment, aucun des individus de la population n'a évolué. En effet, seulle la mutation pourra induire des changements dans nos individus.

*2) **Mutation**:* Le rôle des opérateurs de mutation est certainement le plus important dans notre algorithme. Il va nous permettre une recherche d'individus meilleurs dans l'espace de recherche et étant donné que nous n'utilisons pas de croisements, ils seront les seuls à pouvoir permettre une modification de nos individus courant. Trois types de mutations seront donc utilisés : une recherche locale et un algorithme Greedy pour améliorer principalement le nombre de couleurs, ainsi qu'un dernier permettant un équilibrage des couleurs de notre graphe et donc de notre individu.

*a) **Recherche Locale**:* Le but de cette recherche locale va être d'améliorer le nombre de couleurs de notre graphe et donc de minimiser ce nombre tout en enlevant les conflits qui pourraient apparaître lors de l'attribution d'une nouvelle couleur à un sommet. Voyons le fonctionnement de cet algorithme.

Si un sommet possède comme couleur la couleur maximale qui existe dans ce graphe, alors ce sommet sera colorié avec une nouvelle couleur choisie aléatoirement parmi toutes les couleurs existantes.
Nous bouclerons (1) ensuite pendant un certain nombre maximal d'itérations où nous stockerons les différents conflits existants dans une liste, c'est à dire que tous les sommets ne pouvant être de la couleur qu'ils sont actuellement seront stockés. Si aucun conflit n'est présent dans le graphe, alors nous venons de gagner une couleur et notre boucle, ainsi que la recherche locale, se terminent. En revanche, si des conflits sont présents, alors nous changerons tous les sommets conflictuels par une nouvelle couleur aléatoire tirée dans les couleurs existantes puis nous recalculerons le nombre de conflits existants. Si aucun conflit n'est maintenant présent, alors comme précédemment, nous sortons de notre boucle et la recherche locale est terminée.
Sinon, nous entrons dans une nouvelle boucle (2), elle aussi fixée par un nombre maximal d'itérations, où nous allons choisir un sommet conflictuel au hasard à chaque itération.

Nous assignerons tour à tour chaque couleur existante à ce sommet et regarderons les conflits présents pour ce sommet et uniquement ce sommet. Si aucun conflit n'est présent pour ce dernier, nous pouvons retirer ce sommet de notre liste de conflits et continuer avec un nouveau sommet possédant lui aussi des conflits. En revanche, si ce sommet possède toujours au moins un conflit avec n'importe quelle couleur, alors, nous lui assignerons la couleur pour laquelle le moins de conflits seront présents.

Nous voyons donc, que grâce à cet algorithme de recherche locale, nous cherchons une nouvelle assignation de couleur principalement pour les sommets étant coloriés avec la couleur dite "maximale" dans le but de réussir à retirer cette couleur du graphe et donc à améliorer son nombre chromatique.

*b)* ***Algorithme Greedy (Glouton)****:* Le fonctionnement d'un algorithme glouton est relativement simple. Son but est de faire toujours un choix localement optimal dans l'espoir que ce choix mènera à une solution globalement optimale. Voyons son fonctionnement dans le cas présent.

Nous allons tout d'abord commencer par trier nos sommets. Ce tri sera effectué selon différentes manières : un tri aléatoire ou un tri par ordre croissant ou décroissant, en fonction du degré de chaque sommet. Le tri effectué sera choisi aléatoirement. Nous parcourrons ensuite nos sommets triés et attribuerons à chaque sommet la plus petite couleur possible. Cette couleur devra bien sûr respecter une coloration possible, autrement dit, aucun des voisins du sommet se voyant attribué cette couleur, ne doit avoir la même couleur que ce dernier. Ce procédé sera réalisé plusieurs fois pour chaque graphe, à chaque fois que cet opérateur de mutation sera appelé. Nous obtiendrons donc un nouvel individu avec une nouvelle assignation de couleur.

Contrairement à l'opérateur réalisant la recherche locale, nous voyons ici que cette méthode gloutonne ne cherche pas à enlever une couleur mais à attribuer des couleurs de façon à ce que l'on ai le moins de couleurs possibles.

*c)* ***Équilibrage des classes de couleur****:* Cet opérateur va contribuer à équilibrer le nombre de sommets présents dans chaque classe de couleur. Une classe de couleur est tout simplement représentée par une couleur et les sommets possédants cette couleur.

Nous allons donc tout d'abord commencer par calculer ces différentes classes de couleurs, puis nous réaliserons une moyenne $m$ du nombre de sommets présents dans chaque classe de couleur. Nous trierons ensuite ces classes selon le nombre de sommets qu'elles contiennent dans l'ordre décroissant, puis nous bouclerons dessus dans cet ordre. Si la classe se trouve avoir un nombre de sommets inférieur à $m + 1$, alors nous arrêtons la boucle, sinon nous cherchons dans cette classe de couleur le sommet avec le degré minimal. Une fois ce sommet trouvé, nous cherchons maintenant à le changer de couleur par une nouvelle en commençant cette fois-ci par la fin de notre liste de classes triées, afin d'équilibrer les classes. Si le sommet peut appartenir à la dernière classe (celle possédant le moins de sommets) tout en respectant les contraintes de conflits avec le voisinage, alors nous changeons sa couleur. Sinon, nous essayons avec l'avant dernière classe et nous continuons ainsi de suite jusqu'à trouver une nouvelle classe possible.
Si la nouvelle classe de couleur que nous essayons se trouve posséder un nombre de sommets supérieur à $m - 1$, alors nous arrêtons car nous allons augmenter la taille d'une classe de couleur étant pour le moment à la moyenne calculée ou déjà trop élevée.

Si aucun changement n'est réalisable durant cet algorithme, alors nous choisissons le sommet ayant le plus petit degré de la classe possédant le plus de sommets, et nous changeons la couleur de ce sommet par une nouvelle couleur qui n'existait pas jusqu'à présent, puis nous recommençons l'algorithme précédent.

Nous avons vu, dans cette partie consacrée aux opérateurs de mutation, que certains vont principalement nous permettre de diminuer le nombre chromatique et donc de minimiser le nombre de couleurs présentes dans le graphe alors que d'autres vont nous amener à une coloration équitable ou du moins plus équitable qu'avant. L'alternance de ces opérateurs nous amènera donc vers une solution équitable avec le moins de couleurs possibles.

*3)* ***Insertion****:* L'insertion quant à elle est inspirée sur les mêmes caractéristiques que la sélection. Nous pouvons donc retrouver une insertion en fonction de la fitness des individus présents dans la population et remplacer les plus mauvais, ce que nous ferons ici. Cela permet donc d'avoir une population évolutive et qui, la plupart du temps, améliore la population. Des insertions basées sur l'âge des individus sont aussi trouvées dans la littérature.

De nombreux opérateurs peuvent être imaginés pour l'amélioration du nombre chromatique et de l'équitabilité d'un graphe. Ici, nous en avons vu 3 principaux pour la mutation. Nous allons maintenant devoir faire un choix de l'opérateur, à appliquer tout au long de l'algorithme. Deux possibilités s'offrent à nous : un opérateur fixe tout au long des itérations ou une alternance d'opérateurs. Pour cela nous avons essayé plusieurs méthodes qui vont être brièvement présentées.

## III. CONTRÔLE DES OPÉRATEURS

Le contrôle dynamique se réalise grâce à la sélection d'un opérateur à un moment $t$ dans un ensemble d'opérateurs implémentés. Ces opérateurs pourront être sélectionnés grâces à différentes méthodes. Avant cela, certains algorithmes ont besoin de paramètres afin de pouvoir permettre de sélectionner l'opérateur permettant la meilleure amélioration de fitness d'un individu. Une récompense est donc, par la suite, attribuée à chaque opérateur choisi.
Plus tard, nous considérerons un ensemble d'opérateurs $O = \{o_1, ..., o_n\}$, avec pour chaque opérateur $i$ une récompense associée $r(o_i, t)$ à une itération t.

### A. *Principe de récompense (reward) et utilité*

La récompense, aussi appelée le gain d'un opérateur, se trouve être directement associée à la fitness $f$ d'un individu. L'amélioration qu'aura subi un individu à une itération $t$ sera calculée de la façon suivante : $r(o_i, t) = f_{actuelle} - f_{precedente}$ si $f_{actuelle} - f_{precedente} > 0$, et 0 si la différence et inférieure ou égale à 0 ainsi que si l'opérateur $i$ n'a pas été choisi à l'itération $t$. Ces différentes récompenses seront accumulées au fil des itérations lorsque l'opérateur aura été sélectionné de telle sorte que nous disposons pour chaque opérateur $i \in n$ d'une moyenne de ses récompenses à une itération $T$ de la forme

$$x_i = \frac{1}{T} * \sum_{t=1}^{T} r(o_i, t)$$

Maintenant que nous avons vu le principe de récompense, voyons le calcul de l'utilité, à proprement parler, de notre opérateur. En effet, cette récompense ne sera pas directement utilisée dans nos algorithmes de décision mais permettra de calculer ce qui est appelé utilité dans la littérature. Pour cette dernière, un coefficient $\alpha$ sera nécessaire pour permettre de réaliser une balance entre l'utilité immédiate $u_i(t)$ à l'itération $t$ pour un opérateur $i$, et celle de l'itération précédente $u_i(t-1)$. Nous obtenons alors grâce à notre moyenne des récompenses $x_i$ vu précédemment :

$$u_i(t) = (1 - \alpha) * u_i(t-1) + \alpha * x_i$$

Chaque opérateur verra son utilité recalculée à chaque itération même si celui-ci n'a pas été choisi à cette itération $t$.

### B. *Mécanismes de sélection d'opérateurs*

Maintenant que nous avons pour chaque opérateur $o_i \in O = \{o_1, ..., o_n\}$ une récompense ainsi qu'une utilité, ces valeurs peuvent nous permettre de sélectionner un opérateur. Différentes méthodes existent et nous en présenterons ici seulement deux parmi quatre que nous avons implémentées. Deux parmi elles, la roulette proportionnelle ainsi que l'adaptive pursuit, calculent une probabilité $p_i$ d'être choisi pour chaque opérateur $o_i, \forall i \in \{1, ..., n\}$. Celle dite de l'UCB pour Upper Confidence Bound utilise directement l'utilité calculée et se base comme son nom l'indique sur la confiance de la limite supérieure que nous verrons en dernier point.

*1) **Sélection par Adaptive Roulette Wheel (Roulette Proportionnelle Adaptative)**:* Une nouvelle probabilité sera calculée pour tous les opérateurs, une fois celui-ci appliqué, et donc sa nouvelle récompense déterminée. Dans le but de permettre une sélection à ceux possédants une moyenne de leurs récompenses faibles, un paramètre $p_{min}$ compris entre $[0, \frac{1}{n}]$ sera introduit pour assurer une probabilité minimale à chaque opérateur et donc un minimum d'exploration des opérateurs. Les probabilités calculées des opérateurs pour l'itération $t+1$ étant donné, les nouvelles utilités calculées à l'itération $t$, après avoir appliqué un des opérateurs, sera donc de la forme :

$$p_i(t+1) = p_{min} + (1 - n * p_{min}) * \frac{u_i(t)}{\sum_{k=1}^{n} u_k(t)}, \forall u_i, i \in \{1, ..., n\}$$

Le $i$-ème opérateur est maintenant choisi proportionnellement à sa probabilité. Un "mauvais" opérateur devrait donc être sélectionné avec une probabilité $p_{min}$ alors que le meilleur devrait l'être avec une probabilité $p_{max} = 1 - (n-1) * p_{min}$.

*2) **Sélection par Fixed Roulette (Roulette Fixe)**:* Nous nous sommes rapidement aperçu que l'Adaptive Roulette converge rapidement vers un opérateur entre la Recherche Locale et le Greedy ce qui est une bonne chose pour le début de notre algorithme. En revanche, après avoir obtenu un nombre chromatique plutôt bon, le choix de l'opérateur continue sur ces opérateurs et ne cherche donc pas à équilibrer nos couleurs.

C'est pour cela que nous avons décidé d'implémenter une roulette proportionnelle fixe qui consiste à choisir une probabilité fixe pour chaque opérateur, qui ne varie pas tout au long de l'AG. Chaque opérateur se voit ici attribuer la même probabilité et nous pouvons donc considérer qu'ils seront tous choisis autant de fois pendant notre algorithme. Ceci nous permet donc d'explorer tous les opérateurs, et de pouvoir permettre de rechercher une coloration équitable et la plus minimale possible.

## IV. EXPÉRIMENTATIONS

Comme nous avons pu le voir précédemment, nous gérons le problème de coloration équitable de deux façons. Premièrement nous sommes à la recherche d'une coloration utilisant le moins de couleur possible grâce à notre initialisation puis dans une seconde partie, nous cherchons à équilibrer au mieux cette coloration.

Étant donné que notre algorithme DSATUR ne prend en compte seulement que le nombre de couleurs présentes dans le graphe et non pas l'équitabilité, nous pouvons avoir une première intuition consistant à se dire que nous nous attendons au début à obtenir une fitness élevée, dû à la pénalité que nous ajoutons. Ensuite, notre opérateur permettant d'équilibrer les couleurs, devrait nous permettre d'améliorer rapidement notre individu puis les différents opérateurs que nous avons vu précédemment s'enchaîneront de telle sorte à continuer d'améliorer notre coloration avec le moins de couleurs possibles, tout en essayant d'équilibrer le nombre de sommets dans chaque classe de couleur (Figure 1).

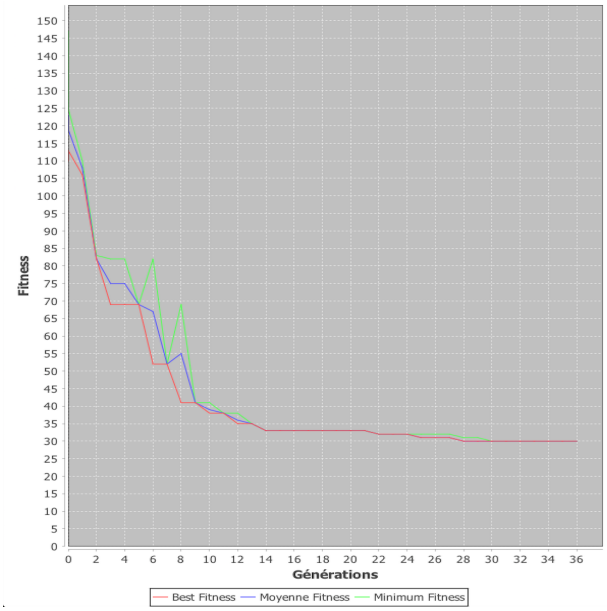


FIGURE 1. Évolution de la fitness des individus en fonction des générations sur le graphe le450_25d

Dans ces expérimentations, nous choisirons de fixer une taille de population égale à 10 et une sélection de deux parents. De même, le critère d'arrêt de notre algorithme sera pour un temps supérieur à 45 secondes. Voyons maintenant nos résultats comparés à ceux bien connus de la littérature.

| Instances | Nos meilleurs résultats | Meilleurs résultats connues |
|---|---|---|
| mulsol.i.2 | 37 | 36 |
| R125.1 | 5 | 5 |
| R250.1 | 10 | 8 |
| R250.5 | 69 | 66 |
| queen8_8 | 10 | 9 |
| school1 | 17 | 15 |
| dsjc125.5 | 20 | 17 |
| dsjc250.5 | 36 | 30 |
| le450_25d | 30 | 26 |
| flat300_28_0 | 40 | 34 |

## V. CONCLUSION

Comme nous avons pu le constater dans le tableau précèdent, nous obtenons des résultats plutôt satisfaisants par rapport aux meilleurs résultats connus dans la littérature. En revanche, lorsque nous avons testé notre algorithme sur des grandes instances possédant beaucoup d'arêtes, nous nous sommes rapidement aperçu que notre conception ne le permettait pas. En effet, une itération prend beaucoup plus de temps et notre algorithme ne peut trouver efficacement une solution.

Une parallélisation du code aurait pû être envisagée mais elle n'était pas l'objectif de cette recherche. De même, des croisements auraient pu permettre une meilleure diversité des individus et donc peut-être un meilleur résultat par la suite. En revanche, comme évoqué précédemment lors de ce papier, lorsque nous avons réalisé nos tests, une correction complète des individus après un croisement ne nous paraissait pas une solution adéquate. L'ajout d'une pénalité supplémentaire dans la fonction d'évaluation pour permettre une mauvaise coloration peut être une des solutions envisagées à l'avenir.

## RÉFÉRENCES


[1] Kosowski Adrian and Manuszewski Krzysztof. *Classical Coloring of Graphs, Chapter 1*.

[2] Lévêque Benjamin. *Coloration de graphes : structures et algorithmes*. PhD thesis, 2007.

[3] Furmanczyk Hanna, Jastrzebski Andrzej, and Kubale‡ Marek. Equitable coloring of graphs. recent theoretical results and new practical algorithms. 2012.

[4] Al Rabat Chowdhury Hasin, Farhat Tasneem, and H. A. Khan Mozammel. Memetic algorithm to solve graph coloring problem. 2013.

[5] T. Karthick. Note on equitable coloring of graphs. 2014.

[6] Lai Xiangjing, Hao Jin-Kao, and Glover Fred. Backtracking based iterated tabu search for equitable coloring. 2015.

[7] Kokosinski Zbigniew, Krzysztofn Kwarciany, and Kolodziej Marcin. Efficient graph coloring with parallel genetic algorithms. 2005.