Optimisation par essaim particulaire

et

Coloriage de graphe

Etude en vue du traitement du problème de l'affectation de fréquences

Version expurgée pour diffusion libre

Juin 2001

Maurice.Clerc@WriteMe.com

Généralités

Informations sur le coloriage de graphe et l'affectation de fréquences

De nombreux sites internet, livres et articles traitent de ces questions. Pour des travaux relativement récents voir par exemple les sites [1-4], la thèse très complète [5] ou l'article [6], qui sont en même temps de bons "points d'entrée" vers d'autres documents. Il faut cependant noter que pour des raisons de secret industriel les meilleurs algorithmes ne sont pas nécessairement publiés.

Informations sur l'OEP

La quasi totalité des informations sur l'OEP (articles, personnes, manifestations, présentations etc.) est disponible sur le site internet *Particle Swarm Central* [7] que je maintiens avec Jim Kennedy, un des deux co-fondateurs de cette méthode (l'autre étant Russ Eberhard). Les articles de présentation de la technique ou de ses variantes sont essentiellement [8-13]. L'analyse mathématique la plus complète, illustrée par une série de tests sur des fonctions classiques, se trouve dans [14]. Les principales applications industrielles tirent partie de la remarquable rapidité de convergence de la méthode dans de nombreux cas, pour l'inclure dans des systèmes hybrides temps réels ou presque (par exemple pour accélérer la phase de (ré)apprentissage de réseaux neuronaux ou piloter des processus industriels) ([15-17]). Enfin, à ma connaissance, mais les choses évoluent très vite, l'OEP n'est inscrite au programme d'enseignement normal que dans deux universités, une au Canada et une en Allemagne ([18 , 19]), bien que des tutoriaux aient lieu ailleurs de temps en temps.

L'essentiel de l'OEP

Etant donnés un espace de recherche et une fonction sur cet espace dont on cherche l'optimum, le principe de base consiste à définir un ensemble de particules (l'essaim) par des positions et vitesses initiales, puis à le faire évoluer selon un processus raisonnement-décision à chaque pas de temps, qui, informellement, est le suivant, du point de vue de la particule :

- je me souviens de ma meilleure position jamais trouvée (meilleure position atteinte)
- je demande à chacun de mes voisins leur meilleure position atteinte
- je ne considère que la meilleure d'entre elles
- je combine trois tendances : ma propension à suivre mon propre chemin (attitude "volontariste"), ma tendance "conservatrice" (revenir vers ma meilleure position atteinte) et ma tendance "suiviste" (aller vers mon meilleur voisin) pour définir ma nouvelle vitesse/direction de déplacement
- j'effectue ce déplacement

La formulation générale de ce comportement est la suivante :

$$\begin{cases} v(t+1) = \lambda_1 v(t) + \lambda_2 (p_i(t) - x(t)) + \lambda_3 (p_g(t) - x(t)) \\ x(t+1) = x(t) + v(t+1) \end{cases}$$

où t est le temps, x la position (multidimensionnelle) de la particule, v sa vitesse, p_i sa meilleure position atteinte, p_g la meilleure des meilleures positions atteintes dans son voisinage et λ_1, λ_2, λ_3 les coefficients de confiance pondérant les trois directions possibles (volontariste, conservatrice, suiviste).

Dans presque toutes les applications, les coefficients λ_2 et λ_3 sont choisis à chaque pas de temps au hasard dans un intervalle donné. On voit tout de suite que d'innombrables variantes sont possibles, y compris la possibilité de jouer sur la taille et la définition des voisinages, ou la taille de l'essaim. Pour autant, tout n'est pas permis. En particulier, l'analyse mathématique que j'ai réalisée en 1999, et en fonction de laquelle Jim Kennedy a sensiblement amélioré ses propres versions ([14]), établit les relations que doivent respecter les coefficients pour assurer la convergence (ou, plus précisément la non divergence). Pour la présente étude, j'utilise une version adaptative évolutive (cf. [20] et la section *Position, vitesse, déplacement*).

Les équations du mouvement sont valables quel que soit l'espace de recherche, pour peu que l'on sache définir les opérateurs nécessaires :

- différence de deux positions (déplacement assimilable à une vitesse, le temps étant discrétisé)
- combinaison linéaire de deux vitesses
- addition d'une position et d'une vitesse

Concernant la fonction à optimiser, la seule contrainte est que l'on puisse comparer les valeurs qu'elle prend en deux points quelconques de l'espace de recherche (structure algébrique de treillis, qui garantit certaines propriétés de convergence [21]).

Etude

NOTE : on suppose ici connue l'optimisation par essaim particulaire adaptative, [...]

Introduction

Ce document présente le bilan provisoire, après cinq mois, d'une tentative d'appliquer l'OEP au problème de l'affectation de fréquences, ou, plus précisément pour l'instant, au problème du coloriage de graphe, formellement similaire [5]. [...] a fourni un jeu d'essai de 18 graphes de 75 à 300 nœuds et j'ai étudié les points suivants :

- application de l'OEP « générale », en recherchant les meilleurs paramétrages (espace de recherche, coefficients),
- développement d'opérateurs spécifiques au coloriage de graphe (initialisation, combinaisons linéaires, confinements à des sous-espaces de coloriages, recherche locale).

[...]
Après avoir présenté les résultats bruts, je détaillerai précisément comment ils ont été trouvés, pourquoi, du strict point de vue de l'OEP, ils ne sont pas très bons et comment on peut espérer les améliorer.

Résultats

Le tableau 1 montre les meilleurs résultats actuellement obtenus. La densité du graphe est le rapport du nombre d'arcs au nombre maximum possible. Pour un graphe sans boucle bidirectionnel de N nœuds, ce nombre maximum est $N(N-1)/2$. Le nombre d'évaluations est le nombre de fois que la fonction à minimiser est évaluée. Le niveau de difficulté est une estimation d'après les résultats indiqués dans [22].

La distinction est très tranchée entre les problèmes « faciles », pour lesquels l'application des opérateur de confinement spécifiques détaillés plus loin donne une solution dès la phase *initialisation + confinement*, et les problèmes « difficiles », pour lesquels, tout en ayant parfois rapidement une solution approchée assez bonne, la solution optimale n'est (peut-être) atteinte qu'au bout d'un très grand nombre d'évaluations. Notons, pour fixer les idées, que le niveau de difficulté 5 correspond à environ une heure de calcul sur une station SPARC 10 exécutant l'algorithme génétique défini dans [...]

Tableau 1. Résultats obtenus. L'astérisque indique que la solution a été trouvée lors du confinement spécifique après initialisation.

Graphe	Nb de nœuds du graphe	Densité du graphe	Nb de couleurs minimum connu	Nb de couleurs trouvé	Empan des couleurs possibles	Nombre d'évaluations	Temps (Pentium II)	Niveau de difficulté
4.75.10	75	10	4	4	[-500,500]	1*	0,11 s	2
4.75.20	75	20	4	4	[-500,500]	1*	0,11 s	2
4.75.30	75	30	4	4	[-500,500]	1*	0,11 s	2
8.75.10	75	10	8	8	[-500,500]	1*	0,11 s	0
8.75.20	75	20	8	8	[-500,500]	1*	0,11s	0
8.75.30	75	30	8	9 8	[-500,500] [-500,500]	1* >100000	0,11 s	1
8.150.10	150	10	8	8	[-500,500]	1*	0,11 s	0
8.150.20	150	20	8	10 9	[-500,500] [-500,500]	1* >100000	0,55 s	5
8.150.30	150	30	8	15 14	[-500,500] [-500,500]	1* >100000	0,27 s 0,22 s	4
15.150.10	150	10	15	15	[-500,500]	1*	0,11 s	0
15.150.20	150	20	15	15	[-500,500]	1*	0,11 s	2
15.150.30	150	30	15	16 15	[-500,500] [-500,500]	1* >100000	0,17 s	4
15.300.10	300	10	15	15	[-500,500]	1*	0,16 s	3
15.300.20	300	20	15	17 16	[-500,500] [-500,500]	1* >100000	3,96 s	5
15.300.30	300	30	15	24 23	[-500,500] [-500,500]	1* >100000	23,9 s	5
30.300.10	300	10	30	30	[-500,500]	1*	0,17 s	2
30.300.20	300	20	30	30	[-500,500]	1*	0,44 s	3
30.300.30	300	30	30	30	[-500,500]	1*	0,77 s	4

Paramétrages et principaux composants du programme

Espace de recherche

Dans toute cette étude, j'ai utilisé une représentation directe, au sens où l'espace de recherche est, a priori, l'ensemble des coloriages possibles du graphe analysé (comme indiqué brièvement à la fin, une approche duale est également envisageable). Pour un graphe G de taille N, un coloriage sera en fait un étiquetage numérique, c'est-à-dire une liste de N « couleurs » (nombres, non nécessairement entiers), affectées aux nœuds. Un coloriage est dit admissible si la différence des couleurs de deux nœuds adjacents a une valeur absolue au moins égale à une valeur donnée (1, pour le cas de coloriage simple étudié ici).

Avec une telle représentation directe, l'espace de recherche est théoriquement infini mais, en pratique, il est nécessaire de limiter au préalable le nombre de couleurs c_{max} à utiliser. D'un côté, il est certain qu'un graphe connexe de taille N peut toujours être colorié de façon admissible avec au plus N couleurs. On peut donc être tenté de prendre comme espace de recherche l'ensemble $H = [1,N]^N$, où $[1,N]$ représente touts les entiers entre 1 et N. Mais d'un autre côté, augmenter la taille de l'espace de recherche permet aussi d'augmenter le nombre de solutions accessibles. En effet, de toute solution, on peut en déduire une infinité d'autres par diverses opérations algébriques de substitution (les plus communes étant la translation et la dilatation. La translation consiste à ajouter la même valeur à toutes les couleurs des nœuds et la dilatation à ajouter une valeur fonction croissante de la couleur, par exemple 0 pour la plus petite, 1 pour la suivante etc.).

Le paramètre important est en fait la densité de solutions. Pour les graphes du jeu d'essai, elle diminue rapidement quand c_{max} augmente. A priori, il semble donc que l'on ait intérêt à avoir l'espace de recherche le plus petit possible.

Densité de solutions

Considérons l'espace de recherche $H_0 = [1,c_{max}]^N$. Si, pour $c_{max} = c_{opt}$, on connait le nombre de solutions $S(c_{opt})$, un calcul combinatoire immédiat nous donne $S(c_{\max}) = S(c_{opt})C_{c_{\max}}^{c_{opt}} = \frac{c_{\max}!}{c_{opt}!(c_{\max} - c_{opt})!} S(c_{opt})$, où c_{opt} est le nombre optimum (minimum) de couleurs possibles dans une solution. En divisant par $c_{\max}^N$, on obtient la densité de solutions. Dans le cas où c_{opt} est supérieur à $N/2$, elle passe par un maximum pour une certaine valeur de c_{max}, ou peut même être croissante, mais, sinon, elle diminue constamment.

Cependant, d'une part la densité de solutions reste très faible dès que les graphes sont un peu importants et, d'autre part, on n'est pas forcément certain que pour toutes les solutions les numéros des couleurs soient consécutifs. De plus, si H_1 est le sous-espace des coloriages admissibles (toutes les contraintes respectées) et H_2 le sous-espace des coloriages minimaux (ayant le nombre de couleurs optimum), il est évident que les solutions sont dans $H_1 \cap H_2$. C'est pourquoi il peut sembler intéressan, au cours du processus de recherche, de confiner les particules dans l'un ou l'autre de ces sous-espaces particuliers,. Mais confinements sont d'autant plus faciles que c_{max} est grand.

Par exemple, la densité de H_1 par rapport à H_0 augmente avec c_{max} . La valeur exacte est un peu compliquée à calculer (elle dépend non seulement du nombre de solutions sur $[1,c_{opt}]^N$, mais aussi de la structure de ces solutions, en termes de nombre de nœuds de chaque couleur), mais on montre facilement qu'elle tend vers 1. Et, rapportée à ce sous-espace, la densité de solutions diminue beaucoup moins vite quand c_{max} augmente.

Ainsi, pour le graphe à 10 nœuds de la figure 1, on a $c_{opt} = 3$, et S(3) = 18. La densité de coloriages admissibles (qui, dans ce cas, sont aussi optimums) est de l'ordre de 304×10^{-6}. Pour $c_{max} = 4$, il y a 18x4 = 72 solutions, ce qui donne une densité de 69×10^{-6}. Mais, par contre, il y a maintenant 3432 coloriages admissibles : dans ce sous-espace, la densité de solutions est de 20979×10^{-6}.

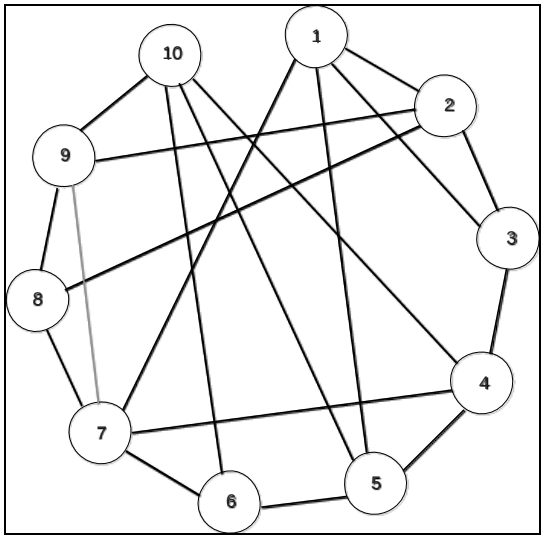


Figure 1. Un graphe 3-coloriable.

Enfin, certains calculs sont d'autant plus longs que l'intervalle des couleurs est grand. Finalement, li y a donc un compromis à trouver. En l'absence, à ce jour, de résultats théoriques permettant d'évaluer la valeur optimale, j'ai procédé par tâtonnements et utilisé $[min, max]^N$, où *min* et *max* sont deux entiers limités par la capacité mémoire de l'ordinateur et, surtout, par le temps de traitement « raisonnablement » acceptable. Typiquement, dans les exemples traités, j'ai pris *min* = -500 et *max* = 500. Les valeurs négatives sont là uniquement pour faciliter la programmation de certains algorithmes. D'un point de vue théorique, cela équivaut à travailler sur [0,1000].

La densité de solutions est en fait la probabilité de trouver une solution par hasard. Donc, inversement, pour un graphe donné, en effectuant de nombreuses fois une recherche purement au hasard, on peut en avoir une estimation statistique. Pour les graphes du jeu d'essai les temps de calcul sur un PC étaient trop prohibitifs pour que j'aie pu faire ces estimations.

Fonction à minimiser

Chaque position dans l'espace de recherche est un coloriage qui peut être caractérisé par deux nombres :

nb_unsat = le nombre de contraintes non satisfaites, c'est-à-dire le nombre de fois que deux nœuds adjacents ont des couleurs dont la différence a une valeur absolue inférieure à 1,
nb_diff = le nombre de couleurs utilisées.

À partir de là, on peut définir classiquement différentes fonctions à minimiser selon que l'on veut privilégier l'un ou l'autre critère. Dans les cas d'école étudiés ici, aux fins de meilleure comparaison, j'ai normalisé les critères sur [0,1] et leur ait affecté le même poids. La fonction choisie est alors

$$f(position/coloriage) = (nb\_diff - c_{opt})/(N-c_{opt}) + nb\_unsat/d_{tot}$$

avec

N = nombre de nœuds du graphe
c_{opt} = nombre minimum de couleurs utilisables (connu, ici)
d_{tot} = nombre d'arcs du graphe

Notons que, de toutes façons, si l'on recherche une solution exacte,la forme précise de la fonction a peu d'importance (ce point a été vérifié sur diverses variantes). Si l'on admettait des solutions approchées, il n'en serait pas nécessairement de même, du moins avec l'OEP adaptative, qui compare quantitativement les positions (l'OEP générale les compare simplement qualitativement).

Initialisation

À l'initialisation, deux particules spéciales sont systématiquement ajoutées, l'une correspondant au coloriage partout nul et l'autre au coloriage « couleur du nœud = nombre de voisins ». Il se trouve en en effet que l'algorithme de confinement à l'ensemble des coloriages admissibles C_0, vu ci-dessous, donne fréquemment directement une solution optimale à partir de l'une ou l'autre de ces particules. Plus de la moitié des exemples traités est ainsi résolue sans faire réellement intervenir les déplacements des particules, c'est-à-dire hors OEP stricto sensu.

Pour les autres particules, l'initialisation est actuellement faite soit totalement au hasard soit selon un algorithme de coloriage partiellement aléatoire :

Coloriage admissible partiellement aléatoire

```
Tant qu'il y a au moins un nœud non colorié
  Choisir au hasard un nœud non colorié
  Construire la liste des couleurs admissibles pour ce nœud, en fonction des nœuds déjà coloriés
  Choisir au hasard une couleur dans cette liste
```

Les vitesses initiales des particules sont toutes nulles, les probabilités que des vitesses aléatoires accélèrent ou ralentissent la convergence étant équivalentes.

Confinements

J'ai utilisé cinq opérateurs (cf. [20]) :

- le confinement d'intervalle (uniquement après un mouvement). Si la particule tend à sortir de l'espace de recherche, elle est ramenée à sa frontière,
- le confinement de granularité. Les couleurs non entières sont remplacées par l'entier le plus proche. Cet opérateur accélère généralement la convergence.
- le confinement spécifique « coloriage admissible » C_0 (respect de toutes les contraintes après initialisation).
- le confinement spécifique « coloriage admissible » C_1 (respect de toutes les contraintes après déplacement).
- le confinement spécifique « coloriage minimum » C_2 (respect du nombre de couleurs souhaitées).

Sauf le premier, ils sont tous optionnels.
Le confinement C_0 est fait selon une adaptation du classique algorithme glouton de coloriage DSATUR [1, 2].

C_0 - Satisfaction de toutes les contraintes et tentative de minimisation du nombre de couleurs

Tant qu'il y a au moins un nœud non modifié
  Choisir le nœud non modifié ayant le plus grand nombre de voisins modifiés de couleurs différentes
    En cas d'égalité, choisir celui ayant le plus grand nombre de voisins
      En cas d'égalité, choisir au hasard
        Affecter la couleur admissible (relativement aux voisins déjà modifiés) la plus proche de la couleur courante (en incrémentant -1, +1, -2, +2 , etc.)

Il est itéré tant qu'il trouve une position ayant moins de couleurs différentes. Il tend ainsi à trouver un coloriage non seulement admissible mais aussi ayant un faible nombre de couleurs. C'est pourquoi il n'est utilisé qu'à l'initialisation, car d'un côté tous les cas simples sont immédiatement résolus, mais, de l'autre, il piège fortement la particule dans un minimum local, et si celui-ci n'est pas un minimum global, cela contrarie trop la recherche ultérieure.

C1 - Satisfaction de toutes les contraintes (au prix de plus de couleurs)

Tant qu'il y a au moins une contrainte non respectée
  Trouver le nœud qui a le plus grand nombre de contraintes non respectées
  Affecter la couleur admissible la plus proche de la couleur courante

C2 - Satisfaction du nombre de couleurs souhaitées (au prix de plus de contraintes non satisfaites)

Tant qu'il y a trop de couleurs
  Trouver la couleur impliquée dans le plus petit nombre de contraintes
  La remplacer par la couleur différente la plus proche déjà utilisée

Les algorithmes C_1 et C_2 peuvent être utilisés séparément ou en alternance. Théoriquement, le rôle de tels algorithmes est d'assurer une « projection » d'un point de l'espace H_0 de tous les coloriages possibles sur le sous-espace H_1 (resp. H_2) de tous les coloriages respectant les contraintes (resp. ayant un nombre de couleurs inférieur ou égal à celui souhaité). En pratique, vu la difficulté de ces opérations, ceux présentés ici ne font pas exactement cela. Il serait donc souhaitable d'en trouver d'autres correspondant mieux au modèle mathématique.

Par exemple, pour la projection sur H_1, une adaptation très simplifiée d'une simulation de repliement de protéine semble intéressante. Dans ce nouvel algorithme, en cours de test, chaque arc du graphe est assimilé à un ressort dont la compression est positive dès que la contrainte sur cet arc n'est pas satisfaite. On se trouve donc devant un système physique dont les minimums d'énergie correspondent à des coloriages admissibles. En le laissant évoluer par petits incréments d'un position donnée (et on voit ici tout l'intérêt de pouvoir utiliser des « couleurs » qui ne soient pas des nombres entiers), on peut espérer trouver le coloriage admissible réellement le plus proche de ladite position. C'est effectivement le cas pour de petits graphes, mais la vérification pour des graphes de plus grande taille est assez délicate.

Position, vitesse, déplacement

On rappelle que les équations du mouvement, dans la version adaptative simplifiée choisie, sont, pour chaque particule :

$$\begin{cases} v(t+1) = \alpha v(t) + \beta(p_g(t) - x(t)) \\ x(t+1) = x(t) + v(t+1) \end{cases}$$

où $x(t)$ est la position à l'instant t, $v(t)$ la vitesse et $p_g(t)$ la meilleure position jamais trouvée dans le voisinage (qui inclut la particule elle-même).

Dans [20] le coefficient β était aléatoire sur $[0,b]$, b étant adapté (comme d'ailleurs α), au cours du processus. En fait, il s'avère que cela est inutile : on peut utiliser directement $\beta = b$, ce qui nous donne un processus déterministe (à l'exception des générations et suppressions de particules) plus efficace que le processus partiellement aléatoire. La formule permettant d'évaluer de combien une particule est meilleure qu'une autre a également été simplifiée, tout en donnant de meilleurs résultats. On se contente de calculer la différence relative entre les meilleures positions atteintes par chacune des particules. Par utilisation récursive de l'OEP sur un jeu d'essai de petits graphes (au plus 30 nœuds), on peut vérifier que les intervalles de variation [0,5,0,9] pour α et [0,7,0,9] pour b donnent de bons résultats. Rappelons que l'adaptation elle-même se fait selon la règle que plus

une particule est meilleure que sa meilleure voisine plus elle suit sa propre voie (augmentation de α et diminution de b) et inversement.

Les objets mathématiques et opérateurs algébriques nécessaires à ces équations doivent a priori être définis spécifiquement pour le coloriage de graphe. Rappelons que ces opérateurs sont les suivants :

- différence de deux positions (dont le résultat est identifiable à une vitesse),
- combinaison linéaire de deux vitesses
- application d'une vitesse à une position

Actuellement, j'ai testé trois séries d'opérateurs :

- les opérateurs vectoriels classiques, en considérant chaque position/coloriage comme un vecteur à N dimensions (N = nombre de nœuds du graphe) et chaque vitesse comme un vecteur numérique donnant les variations de position.
- un opérateur spécifique CL_1 pour la combinaison linéaire de deux vitesses (avec coefficients sur [0,1]), dans l'esprit d'une algèbre de graphes, les autres restant vectoriels.
- un autre opérateur de combinaison linéaire CL_2, défini à partir d'un modification de la métaphore psycho-sociale sous-tendant d'OEP classique, et explicitement conçu pour que la nouvelle position de la particule soit formellement équivalente à un croisement « génétique » de deux autres positions.

Combinaison linéaire CL_1

La multiplication d'une vitesse par un coefficient λ se fait selon le processus suivant, dont le résultat est une vitesse incomplète, non directement utilisable :

Calculer l'entier κ le plus proche de λN (ce sera le nombre de composantes de la vitesse à éviter de modifier)
« Effacer » N-κ composantes de la vitesse, à partir d'un nœud choisi au hasard et en prenant récursivement les voisins des nœuds correspondants à des composantes déjà effacées.

Ensuite, l'addition de deux vitesses incomplètes se fait composante par composante :

Si l'une des composantes est effacée dans une vitesse et pas dans l'autre, prendre cette dernière
Si les deux composantes sont effacées, prendre une valeur au hasard dans l'intervalle des couleurs étudiées
Si aucune composante n'est effacée, prendre au hasard l'une des deux.

Combinaison linéaire CL_2

Soit à réaliser la combinaison $v = \lambda_1 v_1 + \lambda_2 v_2$.
Le « raisonnement » de la particule est ici qu'elle suit d'abord sa propre voie, tout en acceptant de la modifier un peu en fonction du voisinage, sous réserve que cela apporte une amélioration. D'où l'algorithme suivant (d_{max} est le nombre maximum d'arcs incidents à un nœud du graphe à colorier) :

Appliquer « virtuellement » la vitesse v_1 à la position initiale x, ce qui donne la position p_1.
De même avec la vitesse v_2, ce qui donne la position p_2.
Calculer l'entier κ le plus proche de $\lambda_2 N/(\lambda_1 + \lambda_2)$
Retenir $v = v_1$
Au plus d_{max} fois
  Sélectionner au hasard κ nœuds
  Appliquer à la position p_1 les couleurs qu'ils ont dans p_2, ce qui donne la position p'_1
  Si amélioration, retenir $v = p'_1 - x$ (différence vectorielle)

Voisinage

Le voisinage choisi est celui le plus classiquement utilisé, à savoir de type « social circulaire ». Notons que cela signifie que la particule a des voisins imposés et non choisis par elle-même au cours du processus (variante à l'étude). Reste le problème de définir sa taille. À nouveau, j'ai utilisé ici une version d'OEP récursive, sur des graphes plus petits que ceux du jeu d'essai (maquettes). Une taille de voisinage de trois donne de meilleurs résultats que la taille couramment utilisée de cinq. Intuitivement, cela est dû à l'aspect chaotique de la fonction à minimiser : une taille de voisinage petite augmente le nombre de sous-essaims autour de minimums locaux, et donc les chances de trouver parmi eux un minimum global. D'un autre côté, plus un sous-essaim est petit, moins il est efficace. Il est donc normal qu'il existe une taille de voisinage optimale, au moins pour un graphe donné.

Naturellement, en l'absence d'autres informations, le pari est que cette taille est valable pour d'autres graphes que les maquettes.

Taille de l'essaim

Pour l'adaptation continue de la taille de l'essaim, j'utilise la technique décrite dans [20]. Au départ il a la taille minimale égale à celle du voisinage (trois), puis ensuite chaque particule tente de se supprimer ou de générer une autre particule selon les réponses aux questions « existentielles » qu'elle se pose :

- la situation s'est améliorée dans le voisinage. Est-ce grâce à moi ? Sinon je tente de me supprimer.
- je suis la meilleure particule du voisinage. Mais la situation s'est-elle pour autant suffisamment améliorée ? Sinon, je tente de générer une nouvelle particule.

En option, j'ai testé une variante introduite dans [17]. Une « mauvaise » particule n'est pas supprimée, mais simplement déplacée sur la position de sa meilleure voisine, tout en conservant sa vitesse et sa mémoire. Mais Y. Yoshida travaille avec des essaims de taille constante. Ici, puisqu'il n'y a plus de suppression, l'essaim ne peut que croître et il faut imposer un maximum, vite atteint. Les résultats sont alors finalement moins bons qu'avec un essaim de taille variable.

Dans tous les essais, la taille maximale de l'essaim est au plus de l'ordre de 30 à 35 particules. Ceci est dû à deux facteurs. Le premier est que la probabilité de réussite de la génération (resp. de la suppression) d'une particule est calculée comme fonction décroissante (resp. croissante) de la taille de l'essaim. Néanmoins le mode de calcul est tel que, par exemple, pour un graphe de 100 nœuds, ce seul mécanisme devrait amener assez fréquemment à des tailles de l'ordre de 50 particules. Le deuxième facteur est en fait une autorégulation, au sens où de nombreuses particules générées se « suicident » rapidement au bout de quelques pas de temps, comme n'ayant pas contribué à une amélioration significative (rappelons en effet que les adaptations, tant des coefficients que de la taille de l'essaim, ne se font pas à chaque instant, mais seulement à des intervalles de temps réguliers, de manière, précisément, à laisser à chaque particule le temps de « faire ses preuves »).

Un point important, nécessitant probablement des améliorations, est la manière dont une particule est générée. Actuellement, cela est fait exactement comme pour l'initialisation (hasard pur ou algorithme de coloriage partiellement aléatoire). Il est sans doute souhaitable de mieux utiliser l'information recueillie et de générer la nouvelle particule au moins partiellement en fonction de celle qui déclenche cette génération. En langage génétique, ceci équivaudrait à un clonage suivi d'une mutation.

Recherche locale ou la stratégie de l'Arlequin

La recherche locale consiste à regarder « autour » d'une position si une autre n'est pas meilleure. Deux grandes options ont été testées, avec des variantes :

- appliquer provisoirement une petite vitesse aléatoire à la particule et la déplacer virtuellement (méthode d'OEP générale),
- utiliser un algorithme spécifique de gommage/recoloriage.

La méthode générale ne donne pas de très bons résultats. Par contre, l'algorithme de recherche locale par gommage/recoloriage augmente en effet les chances de trouver une solution, tout au moins si l'on n'utilise pas d'opérateurs de confinement spécifiques (cf. la section *Efficace ou non ?*).

Gommage/Recoloriage

Gommer les « mauvais » nœuds
Gommer récursivement les voisins des nœuds déjà gommés jusqu'à un niveau prédéfini et de façon à ce qu'il reste au moins un nœuds colorié
Recolorer les nœuds gommés, selon un algorithme quelconque cherchant à minimiser le nombre de couleurs et les contraintes non satisfaites

Les « mauvais » nœuds sont définis comme étant ceux contribuant le plus au non respect des contraintes. L'algorithme de recoloriage est encore une adaptation de DSATUR (cf. C_0 dans la section *Confinement*).
De façon imagée, chaque particule essaie d'uniformiser son « habit d'Arlequin » en le remplaçant par un nouveau patchwork, plus simple.

Un autre algorithme, avec retour arrière, a été écrit. Son temps de calcul étant rapidement prohibitif, je n'ai pu l'appliquer qu'à de petits sous-graphes (un « mauvais » nœud et ses voisins, par exemple). Dans ces conditions, il ne donne pas de meilleurs résultats que l'algorithme précédent.

Mémorisations et critère d'arrêt

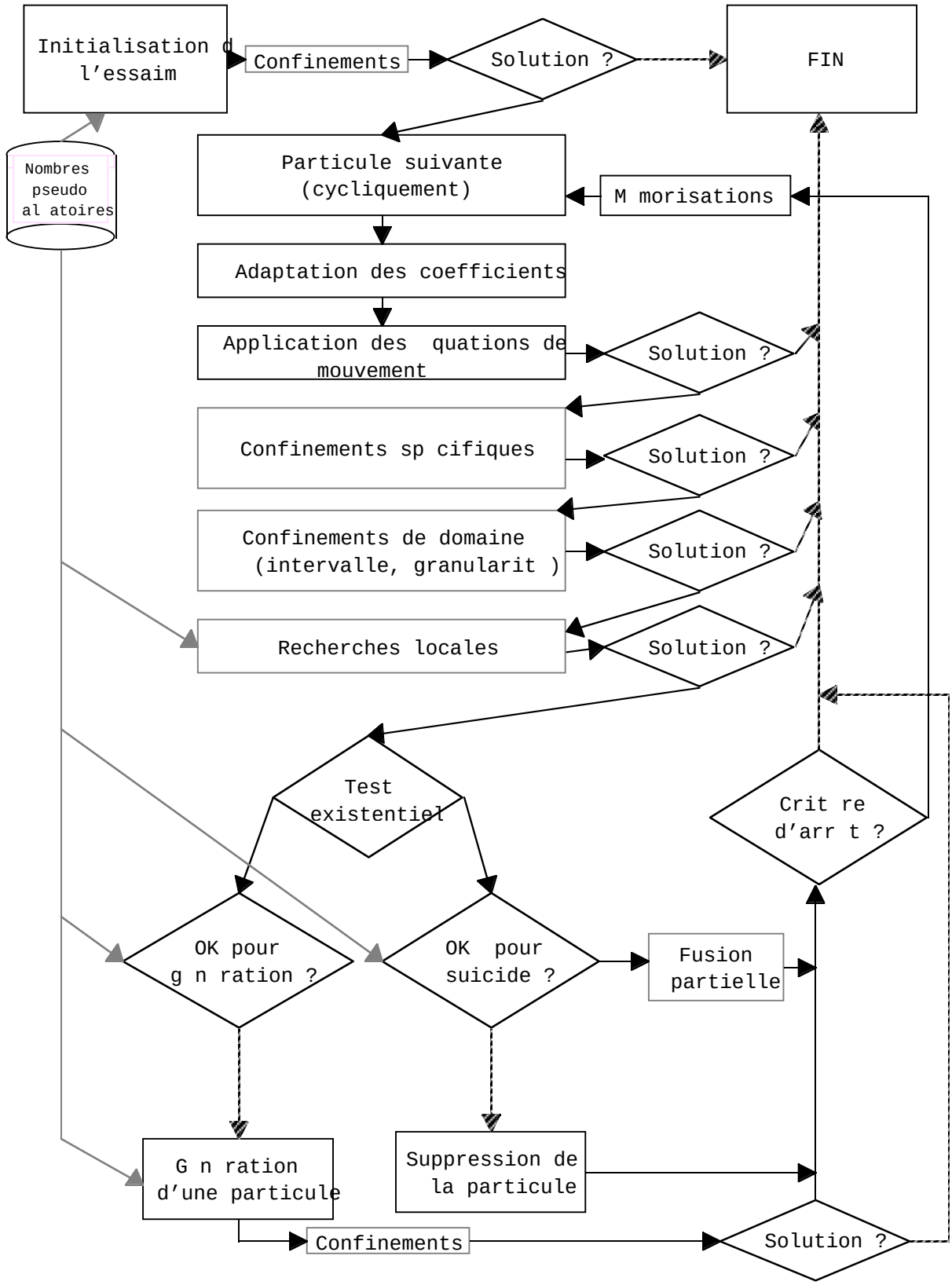
Chaque particule mémorise sa meilleure position jamais atteinte et, aux seules fins de définir un critère d'arrêt automatique, et également autant de valeurs successives de positions antérieures que le voisinage comprend de particules. Ces valeurs servent à estimer un nombre d'évaluations probable pour obtenir une amélioration, pour chaque particule. À chaque pas de temps, on considère le maximum de ces nombres. S'il n'y a pas eu d'amélioration globale malgré un nombre d'évaluations qui lui est supérieur, on considère que la situation est « sans espoir » et la recherche s'arrête. Notons que, pour l'instant, je n'ai pas utilisé de processus *NoHope/ReHope* tel que défini dans [13]. Ce critère semble fonctionner correctement au sens où, si l'on force néanmoins le processus à continuer, on ne trouve en effet pas de solution, même après un très grand nombre d'évaluations.

Reproductibilité des résultats

Un des problèmes pratiques rencontrés était la non reproductibilité parfaite des résultats lors de l'exécution du programme (écrit en ANSI C) sur des machines différentes (Macintosh, PC), du fait que la génération des nombres aléatoires n'est pas nécessairement la même. Seule une reproductibilité statistique pouvait être obtenue.

Ce problème a été résolu en utilisant cycliquement une table de mille nombres quasi-aléatoires sur [0,1] définis une fois pour toute. Tous les essais ont été menés plusieurs fois, en changeant le premier point d'entrée dans cette table, afin de vérifier que cette séquence était suffisamment longue pour ne pas introduire d'artefact.

Ordinogramme du programme



Dans l'ordinogramme simplifié du programme, les modules entourés en pointillés sont optionnels. On notera la fréquence des tests pour voir si, par chance, une solution n'a pas été atteinte. On n'a pas indiqué, car non utilisé, le processus de relance en cas d'échec probable (*NoHope/ReHope*), avec, par exemple, déplacement des particules selon des estimations de gradients de la fonction à minimiser (cf. [13]).

Paramétrage

Tous les paramètres pilotant le fonctionnement du programme ont été volontairement pris identiques pour tous les problèmes. Leurs valeurs ont déjà été indiquées et commentées dans les sections précédentes et le tableau 2 en donne simplement un aperçu synthétique. Par tâtonnement ou exécution récursive, on peut, dans certains cas, trouver de meilleurs jeux de paramètres, mais le bilan global (en temps de calcul) incluant cette recherche préalable est bien supérieur à celui d'une seule exécution directe, même sur un tel jeu de paramètres non nécessairement optimaux.

Rappelons (cf. [20]) que le paramètre « type de traitement » se réfère à la manière dont les particules sont déplacées au cours du temps, soit toutes en même temps au même pas de temps (mode parallèle ou systolique) soit l'une après l'autre, cycliquement. Ce dernier mode, s'est, jusqu'à présent, toujours révélé un peu plus efficace, en termes de nombre d'évaluations nécessaires pour trouver une solution.

Tableau 2. Paramètres d'exécution du programme

$[\alpha_{min},\alpha_{max}]$	[0,5,0,9]
$[b_{min},b_{max}]$	[0,7,0,9]
type de variation de *b*	juste adaptatif (pas d'aléatoire)
taille initiale de l'essaim	3
type d'initialisation	deux particules « spéciales » + particule aléatoire
type de génération d'une nouvelle particule	aléatoire
fusion partielle	non
taille du voisinage	3
type de voisinage	social circulaire
intervalle de variation des « couleurs »	[-500,500], granularité = 1 (nombres entiers)
confinements spécifiques	oui
recherche locale	oui, de niveau 2 (nœud + voisins + voisins de voisins)
type de traitement (séquentiel ou parallèle)	séquentiel cyclique
activation du processus de relance en cas d'échec probable (*NoHope/ReHope*)	non

Efficace ou non ?

Contrairement à ce qui se passe pour d'autres types de problèmes, où les fonctions à minimiser sont moins chaotiques, l'OEP en tant que telle n'est pas très satisfaisante pour le coloriage de graphe.

[...]

D'un autre côté il n'y a pas de toutes façons, à ma connaissance, de méthode d'optimisation combinatoire ou semi-combinatoire déclarée comme étant fondée sur la collaboration dans une population d'agents qui, pour être efficace, ne nécessite en réalité d'utiliser un algorithme de recherche locale puissant même pour une population réduite à un seul agent. En fait, on peut même sérieusement se demander si toutes les opérations d'interaction (croisements en algorithmes génétiques, déplacements vers un "bon" voisin en OEP) ne sont pas tout simplement inutiles. Comme le disent les auteurs en conclusion de [23] :
"*After trying several traditional genetic algorithm variants based on integer and order-based representation we concluded that using only mutation is better than using mutation and crossover*."

Ainsi, si l'on ajoute juste la recherche locale (mais pas les confinements spécifiques), on trouve les résultats du tableau 3 pour les problèmes les plus faciles (notons que dans ce cas, on prend $[1,c_{opt}]^N$ comme espace de recherche).

Tableau 3. OEP « pure » + recherche locale. Résultats pour les problèmes les plus faciles.

Graphe	Nb. d'évaluations pour trouver une solution
4.75.20	28029
4.75.30	34031
8 .75.10	1193
8.75.20	7956
8.150.10	19640
15.150.10	4169
15.150.20	7951
15.300.10	17476
30.300.10	42140

On est loin des performances de l'algorithme de confinement C_0, (rappelons qu'avec lui, une seule évaluation est nécessaire), mais ceci est quand même presque sûrement meilleur que le hasard pur (bien que, comme signalé, je n'aie pu calculer la densité de solutions, vu la taille des graphes).

Conclusion provisoire

[...]

Plusieurs points d'améliorations possibles sont d'ailleurs immédiatement envisageables, en particulier (par ordre décroissant d'importance probable) :

- modification de l'opérateur Combinaison linéaire de vitesses,
- génération plus « intelligente » des nouvelles particules,
- amélioration de la recherche locale, [...]
- utilisation de la fonction *NoHope/ReHope*,
- perfectionnement des confinements spécifiques.

Enfin, comme on l'a vu, toute cette étude est faite sur une représentation directe des coloriages en tant que positions dans un espace de recherche, ce qui pose le problème du choix plus ou moins arbitraire de la taille de cette espace. On peut éviter cet écueil, au prix d'un plus grand nombre de variables, en utilisant une représentation duale, dont je dis ici juste quelques mots.

Approche duale du coloriage de graphe

On considère que les variables à chercher sont, non plus les couleurs des nœuds, mais des pondérations affectées aux arcs. Une pondération d'arcs est admissible si et seulement si, pour tout arc, la valeur trouvée est supérieure à la contrainte sur cet arc (égale à 1 pour un coloriage classique) et si la pondération peut être vue comme la valeur absolue de la différence de couleurs/nombres affectés aux extrémités de l'arc.

Sur cette idée, un correspondant indien (Abhi Dattasharma, de Bangalore) a prouvé le caractère nécessaire et suffisant d'un système d'équations obtenues en parcourant les cycles du graphe. Ce système est à la fois particulièrement grand (autant d'équations, d'ailleurs non indépendantes, que de cycles possibles) et particulièrement simple (équations linéaires à coefficients entiers en nombre réduit, -1 et 1 pour un coloriage classique).

Formellement, les deux problèmes sont évidemment équivalents. Mais alors que le coloriage de graphe en représentation directe se traduit par un système d'inéquations, en représentation duale il donne un système d'équations. Certes, ce système est bien plus grand, mais, par ailleurs, on sait mieux résoudre les équations que les inéquations. Par exemple, c'est précisément le genre de problèmes de très grande taille que les fondeurs de microprocesseurs traitent régulièrement pour la conception de circuits VLSI.

Références

[1] GCRP, http://www.cs.ualberta.ca/~joe/Coloring/index.html#Graph.Colorers

[2] GTHPG, http://www.math.tu-berlin.de/~zuther/math/graph/homes.html

[3] FPRC, http://www.optimization-online.org/DB_HTML/2001/01/251.html

[4] ZIB, http://www.zib.de/

[5] E. Malesinska, "Graph-Theoretical Models for Frequency Assignment Problems," : Technischen Universität Berlin., 1997.

[6] A. Eisenblätter, M. Grötschel, and A. M. C. A. Koster, "Frequency Planning and Ramifications of Coloring," Konrad-Zuse-Zentrum für Informationstechnik Berlin. ZIP-Report OO-47, 2000.

[7] PSC, http://www.particleswarm.net

[8] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," presented at IEEE International Conference on Neural Networks, Perth, Australia, 1995.

[9] R. C. Eberhart and J. Kennedy, "A New Optimizer Using Particles Swarm Theory," presented at Proc. Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995.

[10] P. J. Angeline, "Using Selection to Improve Particle Swarm Optimization," presented at IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4-9, 1998.

[11] A. Carlisle and G. Dozier, "Adapting Particle Swarm Optimization to Dynamics Environments," presented at International Conference on Artificial Intelligence, Monte Carlo Resort, Las Vegas, Nevada, USA, 1998.

[12] Y. H. Shi and R. C. Eberhart, "A Modified Particle Swarm Optimizer," presented at IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4-9, 1998.

[13] M. Clerc, "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization," presented at Congress on Evolutionary Computation, Washington D.C., 1999.

[14] M. Clerc and J. Kennedy, "The Particle Swarm: Explosion, Stability, and Convergence in a Multi-Dimensional Complex Space," *IEEE Journal of Evolutionary Computation*, vol. in press, No. , 2001.

[15] A. Ismail and A. P. Engelbrecht, "Training Products Units in Feedforward Neural Networks using Particle Swarm Optimization," presented at International Conference on Artificial Intelligence, Durban, South Africa, 1999.

[16] S. Naka and Y. Fukuyama, "Practical Distribution State Estimation Using Hybrid Particle Swarm Optimization," presented at IEEE Power Engineering Society Winter Meeting, Columbus, Ohio, USA., 2001.

[17] H. Yoshida, K. Kawata, and Y. Fukuyama, "A Particle Swarm Optimization for Reactive Power and Voltage Control considering Voltage Security Assessment," *IEEE Trans. on Power Systems*, vol. 15, No. 4, pp. 1232-1239, 2001.

[18] U. Ryerson, http://www.ryerson.ca/

[19] IGTE, http://www-igte.tu-graz.ac.at/

[20] M. Clerc, "L'optimisation par essaim particulaire. Principes et pratique," *Hermès, Techniques et Science de l'Informatique*, vol. (soumis), No. , pp. 1-25, 2001.

[21] M. Latapy and H. Duong Phan, "The lattice structure of Chip Firing Games," LIAFA, Université Denis Diderot, Paris 22, 2000.

[22] A. Caminada (1), R. Dorne (2), and J.-K. Hao (2), "Résolution du Problème de l'Affectation des Fréquences par Approche Evolutionniste. Performances sur les Réseaux Fictifs (séries A et B)," (1) France Télécom/CNET Belfort et (2) LGI2P Nîmes, France 1995 1995.

[23] A. E. Eiben, J. K. van der Hauw, and J. I. van Hemert, "Graph Coloring with Adaptive Evolutionary Algorithm," *Journal of Heuristics*, vol. 4, No. 1, 2001.