Recherche tabou générique pour problèmes à contraintes binaires

Alexandre Gondran

Recherche tabou générique pour problèmes à contraintes binaires

Alexandre Gondran
École Nationale de l'Aviation Civile, Toulouse, France
alexandre.gondran@enac.fr

septembre 2014

Résumé

Cette note présente un algorithme de recherche tabou pour les problèmes d'affectation à variables discrètes et à domaines finis et n'ayant que des contraintes binaires. L'intérêt de cette note est de présenter en détail les structures de données et les algorithmes de maintien incrémental des ces structures, clés de la performance des algorithmes à recherche locale. Une série d'application issue du domaine aérien est présenté.

Table des matières

On présente un algorithme de recherche tabou pour les problèmes d'affectation à variables discrètes et à domaines finis et n'ayant que des contraintes binaires. Cet algorithme est une généralisation de TabuCol [11] [1], une recherche tabou performante pour le problème de coloration de sommets d'un graphe. [7, 5] présentent TabuCSP un algorithme plus générale résolvant à l'aide d'une recherche tabou générique le problème MAX-CSP, problème de minimisation des contraintes d'un problème CSP (Constraint Satisfaction Problem, problème de satisfaction de contraintes). Cependant ils ne donnent pas les algorithmes du maintient incrémental des structures de données. Voir plutôt [8] pour le problème de coloration de graphe et [4] qui en donne quelques exemples, malheureusement on trouve difficilement cet article.

1 Problèmes à contraintes binaires

1.1 Variables de décision

— Le problème comporte n variables de décision noté x_i avec $i \in I = [\![1; n]\!]$.
— $\forall i \in I$, le domaine de la variable x_i est noté $D_i = \{d_1, ..., d_{m_i}\}$; il corresponds à l'ensemble des m_i valeurs qui peuvent être affectée à x_i. On note $K_i = [\![1; m_i]\!]$ On note $m = \frac{\sum_{i \in I} m_i}{n}$ la taille du domaine moyen des variables.
— Une solution du problème est notée $S = (s(i))_{i \in I}$ avec $s(i) \in K_i$ où la $s(i)$-ème valeur du domaine D_i est affectée à la variable x_i.

1.2 Données du problème

— Pour toutes paires de variables (i, j), on définit la fonction

$$\begin{array}{rccc} \delta_{(i,j)} : & K_i \times K_j & \rightarrow & \{0,\ 1\} \\ & (k, l) & \mapsto & \delta_{(i,j)}(k, l) \end{array}$$

où $\delta_{(i,j)}(k, l)$ indique si les variables i et j sont en conflit lorsqu'elles prennent respectivement les k-ième et l-ième valeurs de leur domaine. Les résultats de ces fonctions peuvent être stockées dans une matrice $0-1$ 4D abusivement noté $\Delta = (\delta(i, j, k, l))_{I^2 \times K_i \times K_j}$.
— $cost(i, k)$ vecteur indiquant le coût d'affectation de la variable i à la valeur k.

1.3 Objectifs et contraintes du problème

Trouver $S = (s(1), s(2), ...s(n))$ qui minimise :

$$f(S) = \sum_{i \in I} cost(i, s(i))$$

sous les contraintes : $\forall (i, j) \in I^2, \quad \delta(i, j, s(i), s(j)) = 0$

1.4 Remarque

La k-coloration de graphe est un cas particulier de ce problème mais sans objectif et où les variables sont les sommets du graphe et les couleurs le domaine des variables

1. la version de [8] souligne l'importance des structures de données utilisées et du maintient incrémental de ces structures et fournit également des indices algorithmiques.

$(\forall i K_i = K)$ avec :

$$\forall (i,j) \in I^2, \forall (k,l) \in K^2, \quad \delta(i,j,k,l) = \begin{cases} 1 & \text{si } i \text{ et } j \text{ sont deux sommets voisins et } k = l \\ 0 & \text{sinon} \end{cases}$$

$$\forall i \in I, \forall k \in K, \quad cost(i,k) = 0$$

2 Algorithmes de recherche locale

Il y a trois éléments pour définir une recherche locale :

1. L'espace de recherche, c'est à dire la fonction objectif que l'algorithme va réellement optimiser (minimiser). On transforme le problème en relaxant les contraintes.
   Trouver $S = (s(1), s(2), ...s(n))$ qui minimise :

   $$f(S) = (f_1(S),\ f_2(S))$$

   avec :
   — $f_1(S) = \sum_{(i,j)\in I^2;\ i<j} \delta(i,j,s(i),s(j))$ qui dénombre les conflits dans la solution S, c-à-d la violation des contraintes.
   — $f_2(S) = \sum_{i\in S} cost(i,s(i))$ qui représente le coût réel d'affectation des variables.
   On considère l'ordre lexicographique entre $f_1(S)$ et $f_2(S)$.
2. le voisinage considéré, c'est-à-dire les mouvements possible de la solution courante vers une autre solution. Ici, le voisinage considéré est le changement de la valeur d'une seule variable ; il est donc caractérisé par un couple d'entiers (i,k) avec $i \in I$ et $k \in K_i$.
   On note $S_{i\to l} = S\backslash\{(i,s(i))\} \cup \{(i,l)\}$, la solution S dont on a modifié la variable i en la valeur l. On a donc :
   $\mathcal{V}(S) = \{S_{i\to l},\ \forall i \in I, \forall l \in K_i\backslash\{s(i)\}\} = \{(i,k),\ \forall i \in I, \forall k \in K_i\backslash\{s(i)\}\}$
   car $S \notin \mathcal{V}(S)$.
3. La stratégie de recherche : recherche tabou, recuit simulé, descente simple...
   On présente dans la suite un algorithme de descente simple et un algorithme de recherche tabou.

2.1 Algorithme de descente simple

L'algorithme de descente simple (algorithme 1) part d'une solution initiale et l'améliore itérativement en remplaçant la solution courante par la meilleure solution de son voisinage. Il s'arrête lorsque aucune des solutions du voisinage de la solution courante n'apporte pas d'améliorations de la fonction objective.

2.1.1 Matrice de delta-évaluation

Les recherches locales sont plus performantes s'il est possible de réaliser une delta-évaluation ; c'est-à-dire qu'il n'est pas toujours nécessaire de recalculer totalement la fonction objective d'une solution après un mouvement. Soient la solution S et la solution voisine $S_{i\to l}$ qui corresponds à la solution S pour laquelle on a attribué la l-ème valeur à la i-ème variable. Le calcul de la fonction objective de $S_{i\to l}$ est :

Algorithm 1 Méthode de descente simple

Require:

$I = [\![1; n]\!]$ l'ensemble des n variables

$\forall i \in I, K_i = [\![1; m_i]\!]$ l'ensemble des m_i valeurs possibles pour la variable i

$S = (s(1), s(2), ...s(n))$ une solution initiale, $s(i) \in K_i$

Ensure:

S une solution localement minimale

Calculer de E, les matrices de delta-évaluation ; $E(i, k)$: gain de la fonction objectif si la variable i choisi la valeur k à partir de la solution S.

while $\exists E(i, k) > 0$ **do**

$(i^*, k^*) \leftarrow \arg \max_{i \in I, k \in K_i} E(i, k)$

$s(i^*) = k^*$

$f(S) \leftarrow f(S) - E(i^*, k^*)$

Mettre à jour les matrices E

end while

$f(S_{i \to l}) = f(S) - E(i, l)$. Il suffit donc de calculer $E(i, l)$; on définit alors la matrice de delta-évaluation E à n lignes tel que la ligne i a m_i colonnes. En réalité on a deux matrices de delta-évaluation correspondant au deux fonctions à optimiser.

$$E = (E_1,\ E_2)$$

On remarque que :

$$E_1(j, l) = f(S) - f(S_{j \to l}) = \sum_{i \neq j} \delta(i, j, s(i), s(j)) - \delta(i, j, s(i), l)$$

L'algorithme 2 est une procédure de calcul de la matrice de delta-évaluation E_1, sa complexité est en $O(n^2 m)$. L'algorithme 3 est une procédure de mise à jour de la matrice de delta-évaluation E_1, sa complexité est en $O(nm)$. En effet la mise à jour de $E_1(j, l)$ après la modification de S vers $S_{i^* \to k^*}$ donne :

$$\begin{aligned}
\forall j \in I \backslash \{i^*\}, \forall l \in K \backslash \{s(j)\}, \quad E_{1i^* \to k^*}(j, l) &= f(S_{i^* \to k^*}) - f(S_{i^* \to k^*; j \to l}) \\
&= E_1(j, l) + \delta(i^*, j, k^*, s(j)) - \delta(i^*, j, k^*, l) \\
&\quad -\delta(i^*, j, s(i^*), s(j)) + \delta(i^*, j, s(i^*), l) \\
E_{1i^* \to k^*}(j, s(j)) &= 0 \\
\forall k \in K, \quad E_{1i^* \to k^*}(i^*, k) &= d(i^*, k) - d(i^*, k^*)
\end{aligned}$$

L'initialisation de la matrice E_2 est en $O(nm)$ car :

$$\forall i \in I,\ \forall k \in K_i, \quad E_2(i, k) = cost(i, s(i)) - cost(i, k)$$

Pour la mise à jour de E_2 la modification de S vers $S_{i^* \to k^*}$ est en $O(m_{i^*})$ soit en moyenne $O(m)$ car :

$$\forall k \in K_{i^*}, \quad E_{2i^* \to k^*}(i^*, k) = E_2(i^*, k) - E_2(i^*, k^*)$$

Algorithm 2 Calcul de la matrice de delta-évaluation E_1

Require:

$I = [\![1; n]\!]$ l'ensemble des n variables

$\delta(i, j, k, l)$ matrice booléenne indiquant si les variables i et j sont en conflit s'ils prennent respectivement les k-ième et l-ième valeurs de leur domaine.

$S = (s(1), s(2), ...s(n))$ une solution

Ensure:

E_1 la matrice de delta-évaluation

$E_1(i, k)$ gain en nombre de conflit si on affecte à la variable i sa k-ième valeur à partir de la solution S.

$conflit$ la matrice des conflits ; $conflit(i, j) = 1$ si les variables i et j sont en conflit dans la solution S.

for $i \in I$ **do**
  $E_1(i, s(i)) \leftarrow 0$
  for $k \in K_i \backslash \{s(i)\}$ **do**
    $E_1(i, k) \leftarrow 0$
    for $j \in I \backslash \{i\}$ **do**
      if $\delta(i, j, k, s(j)) = 1$ **then**
        if $\delta(i, j, s(i), s(j)) = 0$ **then**
          $conflit(i, j) \leftarrow 1$ // ajout d'un nouveau conflit entre i et j
          $E_1(i, k) - -$
        end if
      else
        if $\delta(i, j, s(i), s(j)) = 1$ **then**
          $conflit(i, j) \leftarrow 0$ // suppression du conflit entre i et j
          $E_1(i, k) + +$
        end if
      end if
    end for
  end for
end for

2.2 Algorithme Tabou

La descente simple s'arrête lorsque tous les voisins de la solution courante sont plus mauvais que celle-ci. On a alors atteins un minimum dit local car il dépends à la fois de la solution initiale et du type de voisinage choisis. Il y a plusieurs méthodes pour sortir de ce minimum locale et pour continuer à explorer d'autres parties de l'espace des solutions. L'une des plus évidentes consiste à choisir un nouveau point de départ et de recommencer, c'est le principe des méthodes muti-départs. On peut également un fois bloqué dans le minimum locale changer de type de voisinage, en l'augmentant par exemple, et ainsi la solution courante n'est plus un minimum pour ce nouveau voisinage, les recherches à voisinages variables [10] utilisent cette idée. On peut aussi modifier la fonction objectif comme le fait le recuit simulé [2].

Pour sortir du minimum, la recherche tabou, proposée en 1986 par Glover [9] auto-

2. En effet on peut considérer le critère d'acceptation du recuit simulé : *aléa* $< e^{-\frac{f(S_{voisin})-f(S)}{T}}$ comme une modification de la fonction objective auquel on ajoute un terme aléatoire positif : $f'(S_{voisin}) = f(S_{voisin}) + T\ln(\textit{aléa}) < f(S)$

Algorithm 3 Mise à jour de la matrice de delta-évaluation E_2

Require:

$I = [\![1; n]\!]$ l'ensemble des n variables.

$K = [\![1; m]\!]$ l'ensemble des m valeurs possibles pour la i-ème variable.

$\delta(i, j, k, l)$ matrice booléenne indiquant si les variables i et j sont en conflit s'ils prennent respectivement les k-ième et l-ième valeurs de leur domaine.

$S = (s(1), s(2), ...s(n))$ l'ancienne solution.

(i^*, k^*) mouvement effectué : la variable i^* prends la k^*-ième valeur.

$k' = s(i^*)$ ancienne valeur de la variable i^*.

E_1 la matrice de delta-évaluation

Ensure:

E_1 la matrice de delta-évaluation mise à jour

for $k \in K$ **do**
  $E_1(i^*, k) \leftarrow E_1(i^*, k) - E_1(i^*, k^*)$
end for
for $j \in I \backslash \{i^*\}$ **do**
  for $l \in K \backslash \{s(j)\}$ **do**
    $E_1(j, l) = E_1(j, l) + \delta(i^*, j, k^*, s(j)) - \delta(i^*, j, k^*, l) - \delta(i^*, j, k', s(j)) + \delta(i^*, j, k', l)$
  end for
end for

rise, quant à elle, de dégrader momentanément la solution courante. Cela permet d'explorer plus tard une nouvelle partie de l'espace de solutions. Plus précisément lorsque la solution courante est dans un minimum locale, c'est-à-dire que tous ces voisins sont plus mauvais qu'elle, on choisit comme nouvelle solution courante le moins mauvais voisins. On dégrade donc la solution courante et pour éviter les bouclages on s'interdit de visiter une solution déjà rencontrée. En effet si on a choisi le moins mauvais des voisins, à l'itération suivante on risque fort de revenir sur l'ancienne solution. On appelle liste tabou la liste des solutions déjà visitées et on s'interdit de les visiter de nouveau. Cette liste est de taille finie et de type FIFO (First In First Out). La taille est appelé durée tabou. Le critère d'arrêt de la recherche tabou peut être varié : durée temporelle (ou nombre d'itérations) à ne pas dépasser ou encore nombre maximal d'itérations sans amélioration. De plus il est souvent coûteux en temps de calcul de vérifier si une solution voisine est dans la liste tabou ; alors la liste tabou n'enregistre pas toutes les solutions déjà visitée mais uniquement les mouvements inverses des mouvements déjà effectués. En effet cela suffit à ne pas retomber sur une solution déjà visitée. La version suivante de recherche Tabou (algorithme 4) est en $O(nmq)$ avec q nombre d'itérations (dès que $q > n$). En effet rechercher le maximum de E est en $O(nm)$ dans le pire cas comme la mise à jour des matrices E.

On présente quelques améliorations et détails de l'algorithme tabou dans les paragraphes suivants.

2.2.1 Critère d'aspiration

La recherche tabou est une correction de la méthode de descente simple mise en place pour sortir des minima locaux. La liste tabou permet d'éviter que l'algorithme boucle (i.e. revienne sur une solution déjà visitée). Étant donné que pour des raisons

Algorithm 4 Algorithme Tabou

Require:

$I = [\![1; n]\!]$ l'ensemble des n variables.

$K = [\![1; m]\!]$ l'ensemble des m valeurs possibles (pour les variables).

$\delta(i, j, k, l)$ matrice booléenne indiquant si les variables i et j sont en conflit s'ils prennent respectivement les k-ième et l-ième valeurs de leur domaine.

S une solution

$iter_{max}$ nombre d'itération maximal

d durée tabou (correspond à un nombre d'itérations)

Ensure:

S^* la meilleure solution rencontrée

f^* la valeur de la fonction objectif pour S^*, i.e. $f^* = f(S^*)$

$iter \leftarrow 1$: itération courante.

$\forall i \in I,\ \forall k \in K,\ T(i, k) \leftarrow 0$: initialisation de la liste Tabou

Calculer $f(S)$

$S^* \leftarrow S,\ f^* \leftarrow f(S)$

Calculer des matrices E

Calculer les variables critiques J (i.e. impliquées dans un conflits)

while $iter < iter_{max}$ **do**

Faire évoluer la solution S vers la meilleure solution non Tabou de son voisinage ; i.e. choisir le meilleur mouvement voisin (i^*, k^*) non Tabou de S :

$(i^*, k^*) \leftarrow \arg \max_{i \in J, k \in K_i} \{E(i, k) \mid i \neq s(i),\ T(i, k) \leqslant iter\}$

$s(i^*) \leftarrow k^*$

$f(S) \leftarrow f(S) - E(i^*, k^*)$

if $f(S) < f^*$ **then**

$S^* \leftarrow S,\ f^* \leftarrow f(S)$

end if

Mettre à jour la liste Tabou T : $T(i^*$, ancienne valeur de $i^*) = iter + d$

Mettre à jour des matrices E

Mettre à jour les sommets critiques J

$iter \leftarrow iter + 1$

end while

de performance, la liste tabou ne contient pas toutes les solutions visitées mais seulement une liste de mouvements interdits, il est possible que l'un de ces mouvements permettent de trouver une solution meilleure que la meilleure solution jusqu'à présent rencontrée par l'algorithme. On s'autorise, dans ce cas, à transgresser le tabou (i.e. effectuer le mouvement tabou). Cette situation se présente lorsque $f(S) - E(i,k) < f(S^*)$, où S^* est la meilleure solution rencontrée jusqu'à présent, S est la solution courante et (i,k) est le mouvement tabou. On appelle critère d'aspiration ce type de transgression ("détaboufication").

Pour mettre en place ce critère d'aspiration, il suffit de remplacer dans l'algorithme 4 la ligne :
$(i^*, k^*) \leftarrow \arg \max_{i \in J, k \in K_i} \{E(i,k) \mid i \neq s(i),\ T(i,k) \leqslant iter\}$
par la ligne :
$(i^*, k^*) \leftarrow \arg \max_{i \in J, k \in K_i} \{E(i,k) \mid i \neq s(i), (T(i,k) \leqslant iter \ ||\ f(S) - E(i,k) < f^*)\}$

2.2.2 Mouvements et sommets critiques

La taille du voisinage (nombre de voisins d'une solution) est égale à $(\sum_{i=1}^{n} m_i) - 1 = n \times m - 1$, la solution courante n'étant pas un voisin d'elle même. Ce voisinage peut compter beaucoup de solutions-voisines ayant la même fonction objectif et qui diffèrent sur des variables peut intéressantes, c'est-à-dire des variables qui sont peu contraintes et peuvent prendre beaucoup de valeur différente. L'idée de [6] est de réduire le voisinage aux seuls variables impliquées dans une contrainte non satisfaite. Ils ont montré que cela accélère grandement les temps de résolution de l'algorithme.

On reprends cette idée ici. Un mouvement (i,k) est dit critique si est seulement si la variable i est impliqué dans un conflit (contrainte non satisfaite). La variable i est également qualifiée de variable critique. L'algorithme Tabou présenté (algorithme 4) réduit le voisinage uniquement aux mouvements critiques. On note J ce sous-ensemble de I : $J = \{i \in I \mid \exists j \in I \backslash \{i\},\ \delta(i,j,s(i),s(j)) = 1\} \subset I$

Lorsque toutes les contraintes sont satisfaites, c'est-à-dire $J = \varnothing$, il faut continuer l'optimisation sur le second critère, le voisinage d'une solution ne peut être vide. Dans ce cas on reprend $J = I$.

2.2.3 Durée tabou : taille de la liste Tabou

Dans l'algorithme 4 la liste tabou est la matrice T de n lignes où la i-ème ligne a m_i colonnes (c-à-d la même structure que la matrice de delta-évaluation). À chaque itération le mouvement inverse du mouvement sélectionné $(i^*, \textit{ancienne valeur de } i^*)$ devient tabou, c'est-à-dire interdit pendant un certain nombre d'itération, ce nombre d'itération est appelé durée tabou (tabu tenure en anglais). C'est un paramètre difficile à paramétrer. Au lieu d'avoir une valeur statique arbitraire, il peut être défini comme un pourcentage de la taille du voisinage de la solution courante ; en effet la liste tabou réduit cette taille et il faut s'assurer que la taille du voisinage finale ne soit ni trop grande ni trop petite. Une trop petite ou trop grande durée tabou risque de provoquer des bouclages de l'algorithme sur des périodes plus longues (cf. aux travaux de [3] sur des durées tabou adaptatives après détection de bouclage).

On reprend ici à titre d'exemple la durée tabou dynamique et aléatoire de [8] qui donne d'excellent résultats pour la coloration de graphe et les autres applications présentées à la section 3 ; elle est égale à : $d = L + \lambda F(S)$ avec L un nombre choisi

aléatoirement dans $[0; 9]$, $\lambda = 0,6$ et $F(S) = |J|$, le nombre de variables critiques de la solution courante S.

2.2.4 Autres améliorations

D'autres améliorations peuvent être apportées. Citons deux exemples. La recherche du meilleur mouvement consiste à rechercher le maximum d'une matrice. D'une itération à l'autre la matrice n'est pas complètement différente. Par conséquent sauvegarder le maximum de chaque ligne peut s'avérer un gain de temps dans les recherches successives. De plus sauvegarder le second maximum (n'appartenant pas à la même ligne, i.e. variable) n'alourdit que peut l'algorithme mais peut être utile pour initialiser le maximum lors de l'itération suivante.

3 Applications

3.1 Coloration de graphes : TabuCol

Soit un graphe $G(V, E)$ où V est l'ensemble des n sommets et E l'ensemble des m arêtes et k un nombre de couleurs disponible ($k << n$). La k-coloration de graphe consiste à attribuer à chaque sommet du graphe une couleur des k couleurs de tel sorte que deux sommets adjacents (liés par une arête du graphe) n'aient pas la même couleur. Comme noté en 1.4, le problème de k-coloration peux être modélisé ainsi. On peux réduire les structures de données dans ce cas ainsi que les algorithmes. On se ramène ainsi à l'algorithme TabuCol [11] avec une version plus performante des structures de données décrites dans [8], qui obtient parmi les meilleurs résultats pour une méthode pure (c'est-à-dire non hybridée avec une autre) sur le benchmark DIMACS [12].

3.2 Allocation de niveaux de vol

Dans le domaine du trafic aérien, on définit un conflit potentiel entre deux avions si les deux avions en suivant leur route respective ne respectent pas une certaine distance de sécurité entre eux. On peut décider alors de leur allouer des niveaux de vol différents. Les performances d'un avion sont liées à son niveau de vol, il a donc un niveau de vol de préférence. Les variables de décision sont donc les niveaux de vol des avions et leurs domaines sont les niveaux de vol autorisés c'est-à-dire les sept niveaux de vol autour du niveau de vol de référence (± 3 le niveau de vol de référence). Le problème de satisfaction des contraintes revient à un problème de list-coloring (problème de coloration de graphe où l'on ne peut affecter à chaque sommet du graphe seulement une liste de couleur autorisés et non toutes les couleurs disponibles). À cela s'ajoute un objectif de trouver une solution la plus proche possible des niveaux de vol de préférence des avions. L'article [2] détaille ce problème et évalue l'algorithme tabou générique sur 12 instances de 22553 à 32156 vols comptant 153594 à 731969 contraintes binaires ; cela correspondant à une journée de trafic au dessus de l'Europe. Les résultats sont très satisfaisants.

3.3 Déconfliction de cluster d'avions

Une des taches des contrôleurs aériens est de garantir une certaine distance de sécurité entre avions. Les avions suivent des routes définit à l'avance et les contrôleurs

en-route [3] devant leur écrans de radar indiquent aux pilotes les manœuvres à effectuer pour respecter ces contraintes de distance. Pour aider le contrôleur dans le choix des manœuvres possibles, il est nécessaire de prédire un conflit potentiel (non respecter des distances de sécurité si chaque avion reste sur sa route). Les manœuvres sont souvent latérales c'est-à-dire que l'avion garde son niveau de vol et effectue un virage à droite ou à gauche avant de revenir sur sa route. Il est également possible de dénombrer l'ensemble des manœuvres que peut réaliser un avion. Il y a donc une variable de décision par avion dont le domaine correspondent aux manœuvres possibles. On associe à chaque manœuvre un coût qui dépends de l'allongement de sa route. Pour chaque paire d'avions, on peut prédire les manœuvres incompatibles c'est-à-dire qui ne respectent pas les distances de sécurité. On retrouve ainsi la modélisation d'un problème à variables discrètes et finis et à contraintes binaires. L'article [1] présent en détail ce problème, ainsi que 120 instances du problème avec 5 à 20 avions en conflits simultanément et plusieurs méthodes de résolution : algorithme génétique et propagation par contraintes (PPC). On trouve une comparaison des résultats sur un site de l'ÉNAC [4], l'algorithme tabou générique donne de bon résultats et lorsqu'il est couplé avec un algorithme génétique simple il obtient les résultats optimaux (qu'à prouver la PPC) et complète l'approche PPC pour les instances que la PPC ne résout pas de façon exacte.

3.4 Ordonnancement de l'arrivée d'avions aux abords d'un aéroport

Lors de l'approche d'un aéroport, un avion doit changer sa vitesse de vol pour préparer son atterrissage. Pour ordonnancer les arrivées des avions sur l'aéroport et gérer correctement les distances de sécurité, il est nécessaire de déterminer l'instant de changement de vitesse pour chaque avion ; ceux sont les variables de décision du problème. On a un intervalle de temps pour chaque avion et les valeurs dans cet intervalle sont discrétisées. Chaque avion a un instant de changement préférentiel et on pré-calcule les incompatibilités (liées aux distances de sécurité) entre deux valeurs de changement de vitesse pour deux avions. On se retrouve dans le cas du problème étudié dans cette note.

Références

[1] Cyril Allignol, Nicolas Barnier, Nicolas Durand, and Jean-Marc Alliot. A New Framework for Solving En-Route Conflicts. In USA/Europe Air Traffic Management Research and Developpment Seminar (ATM Seminar), juin 2013.

[2] Cyril Allignol, Nicolas Barnier, and Alexandre Gondran. Optimized Flight Level Allocation at the Continental Scale. In International Conference on Research in Air Transportation (ICRAT), mai 2012.

[3] Isabelle Devarenne. Études en recherche locale adaptative pour l'optimisation combinatoire. PhD thesis, Université de Technologie de Belfort Montbéliard, France, November 2007.

3. Les contrôleurs en-route gèrent le trafic aérien en-route c'est-à-dire l'ensemble du vol hormis les phases de décollage et atterrissage.

4. `http://clusters.recherche.enac.fr/`

[4] Charles Fleurent and Jacques Ferland. Object-oriented implementation of heuristics search methods for Graph Coloring, Maximum Clique, and Satisfiability. In Johnson and Trick [12], pages 619–652.

[5] Philippe Galinier and Jin-kao Hao. Tabu Search for Maximal Constraint Satisfaction Problems. In Third International Conference on Principles and Practice of Constraint Programming (CP97), pages 196–208. Springer, 1997.

[6] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization, 3(4) :379–397, 1999.

[7] Philippe Galinier and Jin-Kao Hao. A General Approach for Constraint Solving by Local Search. Journal of Mathematical Modelling and Algorithms, 3(1) :73–88, 2004.

[8] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. Computers & Operations Research, 33 :2547–2562, 2006.

[9] Fred Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. Comput. Oper. Res., 13(5) :533–549, May 1986.

[10] Pierre Hansen and Nenad Mladenović. A Tutorial on Variable Neighborhood Search. Technical report, LES CAHIERS DU GERAD, HEC MONTREAL AND GERAD, 2003.

[11] Alain Hertz and Dominique de Werra. Using Tabu Search Techniques for Graph Coloring. Computing, 39(4) :345–351, 1987.

[12] David S. Johnson and Michael Trick, editors. Cliques, Coloring, and Satisfiability : Second DIMACS Implementation Challenge, 1993, volume 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Providence, RI, USA, 1996.