Optimisation des schémas d'ouverture des secteurs de contrôle aérien par méthodes classiques et algorithmes génétiques

David Gianazza

Optimisation des schémas d'ouverture des secteurs de contrôle aérien par méthodes classiques et algorithmes génétiques

David Gianazza
LOG (Laboratoire d'Optimisation Globale) CENA/ENAC
7, avenue Edouard Belin 31055 Toulouse Cedex, FRANCE
Email : gianazza@recherche.enac.fr

Résumé: Ce papier introduit plusieurs algorithmes construisant des configurations optimales de secteurs de contrôle aérien, en tenant compte du trafic prévu, des capacités des secteurs, et du nombre de positions de contrôle disponibles.

Mots clés : Genetic Algorithm, Branch&Bound, A^*, Air Traffic Flow Management, Sector Configuration

1 Vocabulaire et acronymes

Secteur **:** l'espace aérien est divisé en petits volumes appelés *secteurs*, contrôlés chacun par une équipe de deux contrôleurs. Une équipe peut avoir sous sa responsabilité plusieurs secteurs regroupés.

Position de contrôle **:** un meuble muni d'équipements de communication et de surveillance (radar, radio, téléphone...), armé par une équipe de deux contrôleurs qui assurent la séparation entre les avions dans le(s) secteur(s) qu'ils contrôlent.

Configuration de secteurs **:** répartition de n secteurs sur k positions de contrôle

Capacité **:** une valeur seuil du flux de trafic entrant dans un secteur ou un groupe de secteurs, au-delà de laquelle celui-ci est considéré comme surchargé.

CFMU **:** Central Flow Management Unit. Son rôle est de s'assurer que le trafic n'excède pas les capacités, en retardant le décollage de certains avions ou en leur faisant emprunter des routes alternatives.

ACC ou ATCC **:** Air (Traffic) Control Center. En français un centre de contrôle du trafic aérien.

Flow management position (FMP) **:** dans chaque centre de contrôle, une cellule de gestion des flux travaille en collaboration avec la CFMU, et élabore entre autres les schémas d'ouvertures de secteurs.

Schéma d'ouvertures **:** une estimation des configurations de secteurs qui seront utilisés le jour suivant (ou celui d'après). Cette estimation tient compte du trafic prévu et des ressouces disponibles (contrôleurs et positions de contrôle).

2 Introduction

La première étape du processus ATFM (Air Traffic Flow Management) est de définir le schéma d'ouverture des secteurs avec un ou deux jours d'avance, afin de déterminer quelles configurations de secteurs seront les plus adaptées au trafic prévu et si des surcharges sont prévisibles.

En Europe, les schémas d'ouvertures sont élaborés par les FMP (Flow Management Position) de chaque centre de contrôle. Pour ce faire, des outils automatisés sont utilisés par les opérateurs : par exemple en France, un opérateur FMP peut choisir une configuration parmi un ensemble pré-défini et la comparer au trafic prévu afin de faire ressortir les dépassements de capacité. Mais en dehors de sa propre expérience, il n'a aucun moyen de savoir si la configuration choisie est la mieux adaptée ou si une autre aurait permis de mieux équilibrer le trafic entre les positions de contrôle. Un inconvénient supplémentaire de la méthode de travail actuelle est qu'elle se limite à un ensemble de configurations pré-définies qui ne représente qu'une toute petite part de toutes les configurations possibles, comme nous le verrons plus loin.

Le présent article décrit deux algorithmes de recherche dans des arbres et un algorithme génétique permettant de construire des configurations de secteurs optimales en fonction des flux de trafic, des contraintes de capacité des secteurs, et éventuellement du nombre maximal de positions de contrôle qu'il est possible d'armer à chaque heure de la journée. Ces algorithmes sont appliqués à un contexte réaliste, en utilisant les données de trafic, de description de l'espace aérien, de nombre de positions, et de capacités enregistrées par les centres de contrôle français.

3 Autres travaux

Le problème de l'optimisation des configurations de secteurs a déjà été abordé dans [DASF95], dans un contexte peu réaliste, et aussi dans [VM01]. Dans [DASF95], un algorithme génétique est utilisé sur un modèle très simplifié d'environnement ATC, pour équilibrer les charges de travail en regroupant des secteurs. Seuls des secteurs convexes sont envisagés et le nombre de positions est une donnée fixée en entrée.

Dans [VM01], des techniques de programmation entière sont utilisées dans un contexte réaliste afin de minimiser la somme des dépassements de capacité en

sélectionnant des *patterns* parmi un ensemble de configurations pré-définies. Un *pattern* est une configuration associée à une période de temps. Le nombre de positions est également un paramètre fixé en entrée, comme dans [DASF95]. Une tentative est faite de considérer le trafic comme une variable, en autorisant des déplacements macroscopiques de charges de trafic le long de l'axe du temps ou d'un secteur à un autre. La convergence de l'algorithme itératif employé dans ce cas n'est pas assurée. Cependant, les résultats sont encourageants au regard de la méthode de travail actuelle.
Le problème d'optimisation présenté dans cet article est sensiblement différent : l'optimum que l'on cherche à atteindre est la configuration pour laquelle le trafic est au plus prés de la capacité, pour chaque secteur ou groupe de secteurs de la configuration. Les surcharges, mais également les sous-charges, sont donc considérées. Par ailleurs, la recherche d'une configuration optimale ne se restreint pas à un sous-ensemble pré-défini, mais explore l'ensembre des configurations que l'on peut obtenir en combinant des secteurs opérationnels.
Le nombre de positions de contrôle est une variable de la fonction de coût que l'on cherche à minimiser. Ce nombre est éventuellement contraint par une limite supérieure, sachant qu'il n'y a pas toujours suffisamment de contrôleurs pour armer toutes les positions nécéssaires.

4 Modélisation

Une "bonne" configuration est une configuration pour laquelle il n'y a pas de dépassements de capacité (ou les plus petits possibles), et pour laquelle la charge de trafic est aussi équilibrée que possible entre les positions de contrôle, tout en armant le moins possible de positions dans un souci d'efficacité. Ceci peut se formuler comme un problème de minimisation.
Définissons une fonction Δ par

$$\Delta(x,t) = workload(x,t) - capacity(x,t)$$

où x est un secteur ou un groupe de secteurs, t est le temps, $workload$ est la charge de trafic (par exemple N avions entrant dans le secteur x entre t et $t+w$, où w est un intervalle de temps choisi), et $capacity$ la valeur seuil de la charge de trafic. La capacité de chaque secteur peut également dépendre de critères spécifiques (actité militaire, etc...)
L'opérateur peut autoriser certaines tolérances autour de la valeur nominale de la capacité. Ces tolérances inférieure l et supérieure u sont prise en compte dans l'évaluation d'une configuration. Pour ceci, définissons les surcharges et sous-charges comme suit :

$$\Delta_{++}(x,t) = \left| \begin{array}{ll} \Delta(x,t) & \text{si } \Delta(x,t) > u \\ 0 & \text{sinon} \end{array} \right.$$

$$\Delta_{+}(x,t) = \left| \begin{array}{ll} \Delta(x,t) & \text{si } 0 \leq \Delta(x,t) \leq u \\ 0 & \text{sinon} \end{array} \right.$$

$$\Delta_{-}(x,t) = \left| \begin{array}{ll} |\Delta(x,t)| & \text{si } l \leq \Delta(x,t) \leq 0 \\ 0 & \text{sinon} \end{array} \right.$$

$$\Delta_{--}(x,t) = \left| \begin{array}{ll} |\Delta(x,t)| & \text{si } \Delta(x,t) < l \\ 0 & \text{sinon} \end{array} \right.$$

Définissons alors les fonctions suivantes :

$N_{pos}(t)$ le nombre de positions de contrôle de la configuration

$$C_{++}(t) = \sum_{x \in config} (\Delta_{++}(x,t))^2$$

$$C_{+}(t) = \sum_{x \in config} \Delta_{+}(x,t)$$

$$C_{-}(t) = \sum_{x \in config} \Delta_{-}(x,t)$$

$$C_{--}(t) = \sum_{x \in config} (\Delta_{--}(x,t))^2$$

Le problème consiste alors à minimiser la fonction de coût suivante :

$$cost_{config} = a.C_{++} + b.N_{pos} + c.C_{--} + d.(C_{+} + C_{-}) \quad (1)$$

tout en respectant la contrainte : $N_{pos}(t) \leq M_{pos}(t)$ où a, b, c, et d sont des coefficients choisis d'ordre décroissant et où M_{pos} est le nombre maximum de positions de contrôle disponibles à chaque heure du jour.
En pratique et afin de mieux affirmer le poids relatif des différents coûts, plutôt que de minimiser $cost_{config}$ on maximisera $eval_{config}$ telle que les k_1 digits de poids fort de $eval_{config}(x,t)$ soient relatifs au coût C_{++}, les k_2 suivants à N_{pos}, et ainsi de suite :

$$\underbrace{xxxxx}_{k_1}\underbrace{xx}_{k_2}\underbrace{xxxxx}_{k_3}\underbrace{xxx}_{k_4}$$

$$\begin{array}{rl} eval_{config} = & 10^{k_2+k_3+k_4} \times N(k_1, C_{++}) \\ & +10^{k_3+k_4} \times N(k_2, N_{pos}) \\ & +10^{k_4} \times N(k_3, C_{--}) \\ & +N(k_4, C_{+} + C_{-}) \end{array} \quad (2)$$

où N est une fonction telle que

$$N(k,C) = \lfloor max(0, (10^k - 1) - C) \rfloor$$

et les exposants k_1, k_2, k_3, et k_4 sont choisis si possible de façon que le terme $(10^k - 1)$ soit un majorant du coût correspondant.

5 Difficulté du problème

La difficulté du problème tient essentiellement à la combinatoire du nombre de configurations que l'on peut construire à partir d'un ensemble de secteurs. Une configuration est une répartition de n secteurs sur k positions de contrôle.
Cherchons le nombre de façons de partitionner un ensemble de n éléments en k sous-ensembles. Si $P(n,k)$ est ce nombre, il vérifie les relations suivantes :

$\forall n \geq 1$
$P(n,1) = 1$ (un seul groupe de n éléments)
$P(n,n) = 1$ (une seule partition avec n groupes d'un élément)
$P(n,k) = 0$ if $k > n$ (on ne peut pas faire plus de n groupes)
$P(n,k) = k * P(n-1,k) + P(n-1,k-1)$ if $1 < k < n$

Le nombre de partitions possibles sera alors :

$$P(n) = \sum_{k=1}^{n} P(n,k)$$

D'un point de vue algorithmique, l'ensemble des partitions peut théoriquement être obtenu en explorant l'arbre des possibilités, quoique cela devienne rapidement infaisable en pratique.
Pour 17 secteurs (Brest ATCC), la formule $P(17)$ donne environ 83 milliards de possibilités. Cependant, cette valeur n'est pas réaliste : bon nombre des partitions trouvées ne pourraient pas être utilisées dans un contexte opérationnel. Par exemple, une partition contenant un groupe dans lequel un des secteurs n'est voisin d'aucun autre secteur du groupe n'est pas valide.
Afin d'estimer la difficulté du problème réel, considérons seulement les secteurs et groupes de secteurs opérationnels définis dans les bases de données des centres de contrôle. L'ensemble des configurations opérationnelles possibles sera obtenu en parcourant un arbre dont les noeuds sont des listes de couples $(g, \mathcal{G})$ (cf figure 1 illustrant le Branch&Bound), où g est un groupe en construction et $\mathcal{G}$ est l'ensemble des groupes opérationnels compatibles avec g dans le contexte de la configuration en construction. Un élément h de $\mathcal{G}$ est "compatible" avec g s'il contient tous les secteurs de g, mais aucun secteurs des autres groupes de la configuration (les autres "g" du noeud). Si un des ensembles $\mathcal{G}$ est vide, il est inutile de continuer l'exploration à partir du noeud considéré : elle ne mènera à aucune configuration valide.
Les méthodes présentées ci-dessus nous permettent de compter le nombre de configurations et d'estimer la difficulté du problème. Les résultats pour les centres français sont présentés dans le tableau 1, qui souligne bien la relation combinatoire entre le nombre de secteurs et le nombre de partitions et de configurations.
Le nombre de configurations opérationnelles que l'on peut construire à partir des secteurs et groupes de secteurs décrits dans les données des centres va de quelques centaines à plus d'un demi-million. Nous pouvons donc espérer qu'une méthode déterministe d'exploration d'arbre nous mène à des configurations optimales en un temps de calcul raisonnable, si l'on s'en tient aux centres considérés et aux groupes de secteurs opérationnels définis.
Cependant, comme ce problème d'optimisation pourrait être envisagé pour de plus grands centres ou avec un plus grand ensemble de combinaisons de secteurs, un algorithme génétique a également été testé, afin de pouvoir trouver des configurations optimales ou presque optimales en un temps borné.

	Nombre de secteurs		Nombre de partitions	Nombre de configs
	élém.	*groupes*		
Aix	24	42	4.4610^{17}	123,965
Bordeaux	22	65	4.4510^{15}	551,032
Brest	17	52	8.2810^{10}	14,832
Paris (Ouest)	11	17	678,570	192
Paris (Est)	12	22	4,213,597	399
Reims	12	17	4,213,597	249

TAB. 1 – *Nombre de partitions et de configurations possibles pour les centres de contrôle français (données 1999)*

6 Algorithmes déterministes

Un algorithme basique : Dans le chapitre précédent, nous avons décrit un algorithme qui construit toutes les configurations opérationnelles. Une méthode basique d'optimisation consiste à évaluer le coût (cf definition 1) de chacune de ces configurations afin de trouver la ou les meilleures d'entre elles. Cette méthode est consommatrice de temps de calcul et de mémoire mais assure l'optimalité des solutions trouvées. Elle sera donc utilisée comme référence pour l'évaluation des autres algorithmes présentés ci-dessous, lorsque c'est possible.

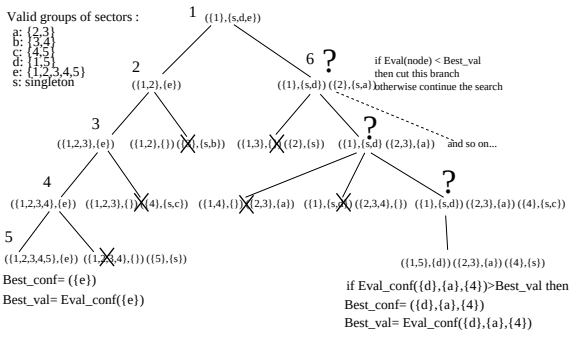

FIG. 1 – *Branch & bound pour la recherche d'une configuration optimale*

Un algorithme de *Branch & bound* **:** L'idée du *branch & bound* illustré sur la figure 1 est d'éviter l'exploration de tout l'arbre en s'interdisant de suivre certaines branches dont on sait qu'elle ne mèneront pas à une meilleure solution que celle qu'on connait

déjà. Pour ceci, il est nécéssaire d'évaluer les noeuds pendant le parcours. Le coût associé à un noeud doit être un minorant du coût de toutes les configurations que l'on peut atteindre à partir de ce noeud. Un noeud de notre algorithme est une configuration en construction (cf figure 1), c'est-à-dire une liste de couples $(g, \mathcal{G})$, où g est un groupe en construction et $\mathcal{G}$ est l'ensemble des groupes opérationnels compatibles avec g dans le contexte de la configuration en construction.

Pour une meilleure compréhension de cette notion de compatibilité, considérons le noeud 6 de l'exemple de la figure 1. Ce noeud est représenté par : $(\{1\}, \{s, d\})$ $(\{2\}, \{s, a\})$. Les groupes compatibles avec le groupe en construction $\{1\}$ sont le singleton s (représentant le secteur $\{1\}$ seul), et le groupe $d = \{1, 5\}$. Ces groupes sont les seuls contenant $\{1\}$ mais pas $\{2\}$.

La fonction de coût pour un noeud est analogue au coût d'une configuration. Soit $best$ une fonction telle que $best(\mathcal{G})$ retourne l'élément h (un secteur ou groupe de secteurs) de $\mathcal{G}$ pour lequel la différence $workload(h, t) - capacity(h, t)$ est la plus petite sous-charge possible, ou alors la plus petite surcharge possible si tous les éléments de $\mathcal{G}$ sont en surcharge. Le coût d'un noeud est alors défini comme dans la définition 1, si ce n'est que N_{pos} est le nombre de couples $(g, \mathcal{G})$ du noeud et que seules les fonctions Δ_{++} et Δ_{+} sont considérées, en prenant $best(\mathcal{G})$ comme paramètre au lieu d'un groupe opérationnel x d'une configuration.

Un algorithme inspiré de l'A^* : Comme le *Branch & bound*, l'A^* est un algorithme de recherche dans un arbre. Cependant, au lieu de mémoriser simplement la meilleure feuille de l'arbre trouvée pendant la recherche, il mémorise les noeuds parcourus en leur affectant une priorité dépendant du côut estimé pour atteindre l'optimum. Pour cela, l'A^* tel que décrit dans [Pea84] nécéssite une fonction calculant le côut de chaque transition d'un noeud à l'autre, et une fonction appelée heuristique, estimant au mieux (en le minorant) le côut des transitions restantes entre le noeud courant et la fin de la recherche.

Pour notre problème, nous cherchons à minimiser la fonction $cost_{config}$ décrite dans la définition 1, en cherchant un chemin dans l'arbre illustré dans la figure 1. Mais au lieu de comparer l'évaluation d'un noeud avec celle de la meilleure configuration trouvée comme dans le *Branch & bound*, les noeuds sont mémorisés dans une *priority queue* ordonnée selon le coût des noeuds. L'A^* considére iterativement le noeud de coût le plus faible, évalue les noeuds fils et les insére dans la queue, ceci jusqu'à ce qu'une configuration complète soit atteinte. Cette configuration est alors optimale.

7 Algorithme génétique

Un algorithme génétique a également été testé pour résoudre le problème d'optimisation des configurations de secteurs. Cet algorithme considère une population de chromosomes qui évolue par croisement, mutation, et selection des individus les plus adaptés, comme décrit dans [Gol89] et [Mic92].

Une configuration est donc considérée ici comme un chromosome composé de plusieur gènes. Un gène est un secteur ou un groupe de secteurs. Les chromosomes sont évalués selon leur adaptation (*fitness*). La valeur brute de l'adaptation d'une configuration est $f = eval_{config}$ (voir définition 2).

Un opérateur de partage, dit *clusterized sharing*, est appliqué aux valeurs brutes de l'adaptation. Le but de cet opérateur est d'éviter qu'un chromosome ayant une bonne adaptation ne se reproduise au détriment des autres chromosomes, focalisant ainsi la recherche sur un seul mode optimal ou sous-optimal. Le *clusterized sharing* modifie les valeurs d'adaptation en fonction de la densité de chromosomes, en identifiant des agrégats (*clusters*) dont on souhaite qu'il évoluent chacun vers un optimum local. Le processus d'agrégation est basé sur un critère de distance entre chromosomes. Pour notre problème, la difficulté du partage consiste à définir une distance entre configurations de secteurs. De façon analogue à la distance de Hamming qui comptabilise les différences entre les gènes de deux chromosomes lorsque le nombre de gènes est une constante fixée, la pseudo-distance choisie pour les configurations se base sur une comparaison des gènes, qui peuvent toutefois être en nombre différent selon les configurations. Soit n_i (respectivement n_j) le nombre de gènes du chromosome i (respectivement j). La pseudo-distance choisie est définie par $d(i, j) = min(n_i, n_j) - n$ où n est le nombre de gènes identiques entre les deux chromosomes.

Les valeurs d'adaptation après partage sont ensuite mises à l'échelle (*sigma truncation*) afin d'atténuer les différences entre les adaptations des bons chromosomes et des mauvais. Ceci laisse une meilleure chance de se reproduire aux mauvais chromosomes, et permet une exploration plus large de l'espace d'états.

L'opérateur de croisement coupe en deux les chromosomes parents et complète chaque morceau (une configuration incomplète) avec les gènes de l'autre parent compatibles avec la configuration incomplète. Une fois tous les gènes possibles de l'autre parent ainsi utilisés, il faut généralement encore compléter la configuration en choisissant au hasard des secteurs ou groupes de secteurs compatibles. Une fonction d'adaptation locale (*local fitness*) permet d'assigner une valeur d'adaptation à chaque gène. Elle est utilisée pour selectionner les gènes des parents en choisissant ceux

dont l'adaptation locale est la meilleure. Il est démontré dans [DA98] qu'un opérateur de croisement utilisant des adaptations locales accélère la convergence dans l'optimisation de fonctions à variables partiellement séparables. Dans notre problème, l'adaptation locale est définie de façon analogue à l'adaptation d'un chromosome, en ne tenant compte des surcharges et sous-charges que du seul gène considéré.
L'opérateur de mutation choisit d'abord un premier gène au hasard. Puis un deuxième choix est effectué dans une liste comprenant le premier gène choisi et ses voisins (les secteurs ou groupes de secteurs de la configuration ayant une frontière commune avec le premier gène). Les secteurs des gènes ainsi choisis sont alors recombinés en plusieurs nouveaux gènes (jusqu'à trois). Ce choix de deux gènes au lieu d'un seul se justifie par le fait que la recombinaison d'un seul groupe se traduirait par un ou plusieurs dégroupements menant ainsi à des configurations plus éclatées. Ce type de mutation n'aurait aucune chance de faire évoluer la population vers des configurations plus regroupées.

8 Resultats

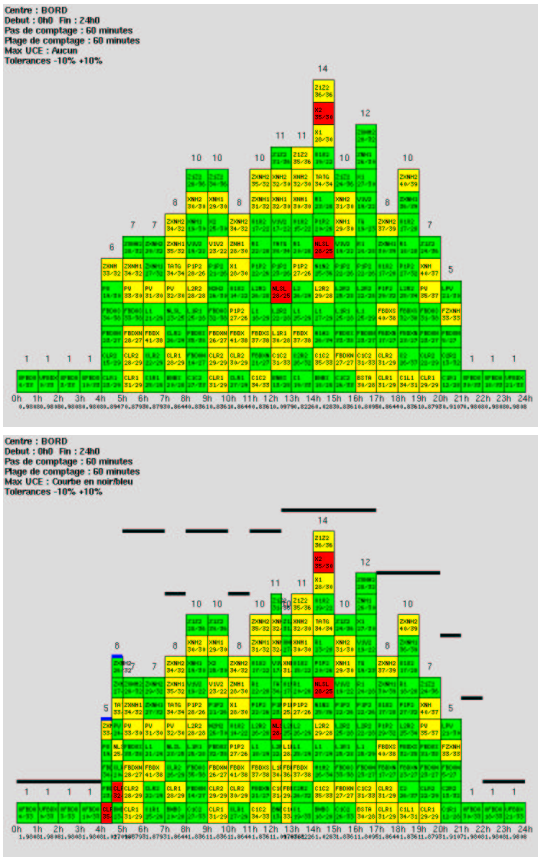

FIG. 2 – *Configurations optimales avec (en bas) ou sans (en haut) contraintes sur le nombre maximal de positions disponibles (exemple de Bordeaux ACC)*

Une interface graphique a été développée afin de visualiser les résultats des optimisations. Les langages Ocaml et OcamlTk ont été utilisés pour coder respectivement les algorithmes et l'interface. Le programme opère sur un PC Pentium IV (1.8 GHz) avec Linux comme système d'exploitation.
Les paramètres d'entrée sont la date, le centre de contrôle choisi, le type de trafic (demande initiale, finale, trafic réalisé), les tolérances sur la capacité, le nombre maximum de positions de contrôle, le pas de calcul et la fenêtre de temps pour le calcul des flux.
Le programme affiche une configuration optimisée pour chaque pas de calcul. Le code de couleur est : Vert (gris) quand la charge est en-dessous de la capacité moins la tolérance inférieure, Jaune (blanc) quand la charge est dans les limites de tolérance de la capacité, Rouge (gris foncé) quand la charge est au-dessus de la capacité plus la tolérance supérieure.
Le haut de la figure 2 montre le résultat d'une optimisation pour le centre de contrôle de Bordeaux, avec en entrée la demande de trafic initiale et aucune contrainte sur le nombre maximal de positions de contrôle. Dans ce cas, les secteurs surchargés ne peuvent être que des secteurs élémentaires ou des groupes de secteurs ne pouvant pas être dégroupés [1].
Le bas de la figure 2 montre la même optimisation en rajoutant des contraintes en nombre maximum de positions disponibles. Ces contraintes sont prises en compte dans les fonctions d'évaluation en diminuant fortement la valeur d'évaluation lorsque le nombre de positions de contrôle de la configuration est supérieur au maximum disponible. Les contraintes de l'exemple de la figure 2 sont issues directement des schémas déposés par le centre de Bordeaux ce jour-là. Ces contraintes induisent des dépassements de capacités supplémentaires (entre 4 et 5 heures du matin sur l'exemple) sur des regroupements de secteurs que l'algorithme ne peut dégrouper par manque de positions disponibles.
On constate en comparant ces deux schémas que Bordeaux était sous-capacitif aux premières heures du matin, et sur-capacitif le reste de la journée.

8.1 Comparaison des algorithmes

Les algorithmes classiques fournissent des solutions optimales pour chaque centre, vérifiées en utilisant l'algorithme basique lorsque c'est possible (pour Brest, Paris et Reims). Le tableau 2 fait état des temps de calcul pour les algorithmes déterministes. Le plus rapide est le *branch & bound*. L'A^* est plus lent car l'heuristique choisie n'est pas suffisamment proche du coût réel des transitions menant à la configuration optimale, ce qui induit un fort taux de backtracking.
Bien qu'il y ait plus de configurations possibles pour Bordeaux que pour Aix (voir tableau 1 sur la difficulté du problème), le temps de calcul pour Aix est

1. certains secteurs élémentaires ne peuvent être armés seuls. Ils sont définis et utilisés uniquement dans un objectif de flexibilité, en choisissant la façon de les combiner avec d'autres secteurs.

	Aix	Bord.	Brest	Paris(E)	Paris(O)	Reims
Basique	n.t.	n.t.	14.44	0.06	0.03	0.02
B&B	34.06	6.06	1.45	0.08	0.07	0.04
A^*	134.88	10.12	10.05	0.08	0.04	0.02

TAB. 2 – *Temps de calcul (en secondes) pour les algorithmes déterministes*

plus important étant donné qu'il y a plus de secteurs à Aix qu'à Bordeaux. L'arbre des possibilités est exploré avec une stratégie de recherche en profondeur, et la profondeur de l'arbre dépend du nombre de secteurs.

Pour l'algorithme génétique, 10 racines du générateur aléatoire ont été testées pour Bordeaux, avec une probabilité de croisement de 0.6 et une probabilité de mutation de 0.2. Le tableau 3 fait état pour chaque pas de temps du nombre de configurations différentes de l'optimum. Les lignes $(+1)$ montrent le nombre d'occurences où la configuration trouvée comprenait exactement une position de plus que la configuration optimale. Les lignes $(> +1)$ montrent le nombre d'occurences où la différence est supérieure à une position de contrôle.

Lorsqu'elles ne sont pas optimales, les solutions trouvées sont qualitativement proches de l'optimum.

	Step	0	1	2	3	4	5	6	7	8	9	10	11
220 gen.	fails	0	0	0	0	0	0	0	0	2	0	1	5
120 elem.	+1	0	0	0	0	0	0	0	0	0	0	0	2
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0
220 gen.	fails	0	0	0	0	0	0	0	0	3	0	0	3
130 elem.	+1	0	0	0	0	0	0	0	0	0	0	0	1
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	1
300 gen.	fails	0	0	0	0	0	0	0	0	0	0	0	0
220 elem.	+1	0	0	0	0	0	0	0	0	0	0	0	0
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0

	Step	12	13	14	15	16	17	18	19	20	21	22	23
220 gen.	fails	2	0	1	3	0	1	0	0	1	0	0	0
120 elem.	+1	0	0	1	0	0	1	0	0	1	0	0	0
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0
220 gen.	fails	3	0	2	2	0	0	0	0	1	0	0	0
130 elem.	+1	0	0	2	0	0	0	0	0	1	0	0	0
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0
300 gen.	fails	0	0	1	0	0	0	0	0	0	0	0	0
220 elem.	+1	0	0	0	0	0	0	0	0	0	0	0	0
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0

TAB. 3 – *Résultats de l'algorithme génétique pour Bordeaux ATCC*

L'algorithme génétique est plus lent que les méthodes de recherche dans des arbres, mais fournit plusieurs solutions optimales ou presque optimales. Il n'est pas très significatif de comparer le temps de calcul de l'algorithme génétique avec les temps de calcul des algorithmes déterministes lorsque le problème est facile. Dans de tels cas (la nuit par exemple), l'algorithme génétique effectuera le même nombre de générations quelle que soit la difficulté du problème, alors que le B&B trouvera la solution optimale parmi les toutes premières branches explorées. Le tableau 4 compare pour Bordeaux les temps de calcul pour chaque pas de temps entre 5h et 19h pour 220 générations et 120 éléments.

Step	5	6	7	8	9	10	11
B&B	0.23	0.05	0.17	0.55	0.95	0.21	0.38
GA	15.69	14.88	17.19	16.08	15.37	15.80	15.75
Step	12	13	14	15	16	17	18
B&B	0.54	0.21	0.66	0.59	0.58	0.35	0.56
GA	16.42	15.62	17.41	16.85	15.65	14.98	13.70

TAB. 4 – *Comparaison détaillée des temps de calcul de l'algogène et du B&B pour Bordeaux ATCC*

8.2 Comparaison entre schéma optimal et schéma déposé

Nous avons vu sur la figure 2 que, même sans contrainte sur le nombre de positions, il pouvait rester des secteurs élémentaires surchargés lorsqu'on considérait en entrée la demande initiale (brute) de trafic. Ceci peut être évité en lissant d'abord le trafic en considérant que tous les secteurs élémentaires sont armés. Comparons les stratégies suivantes en termes de délais produits et de ressources utilisées, pour un jour de pointe de 1999 (le 21 mai) :

DEP : allocation de délais sur la base des schémas déposés par les centres de contrôle français, modifiés en fonction des capacités enregistrées dans les base de données des centres, lorque celles-ci sont différentes des capacités déclarées dans les schémas déposés.

OPT.0.0 : il consiste en deux étapes, la première étant une allocation de délais avec tous les secteurs élémentaires armés, la seconde étant l'optimisation du schéma d'ouverture des secteurs en prenant en entrée le trafic lissé par la première étape.

Il faut être prudent dans l'analyse des résultats : en particulier les valeurs des capacités peuvent être légèrement différentes entre le schéma déclaré et les capacités enregistrées dans les bases de données des centres. Ceci est dû à plusieurs facteurs : en 1999 les capacités n'étaient pas enregistrées régulièrement et le jeu de données utilisé ici peut ne pas correspondre à la journée choisie. Par ailleurs, pour l'élaboration du schéma déposé, l'opérateur choisit parmi plusieurs valeurs de capacité en fonction du contexte du jour (activité militaire, etc...). Quoi qu'il en soit, tout ceci ne change pas la nature et la validité des résultats présentés.

Une méthode ("SHAMAN") d'allocation de délais par programmation par contraintes a été utilisée. Elle minimise le délai maximum. Pour plus de détails sur cette méthode, le lecteur peut se rérérer à [BBR01]. Les allocations de créneaux comme les optimisations de schémas se font sur la base d'un pas de calcul de 60 mn et d'une fenêtre temporelle de 60 mn pour le calcul des flux, et avec des tolérances nulles sur les valeurs des capacités. Les délais générés par chaque méthode sont présentés dans le tableau 5, et les res-

sources utilisées dans chaque cas sont détaillées pour chaque centre dans le tableau 6.

	Retards générés (en minutes)	
	Total	Max
DEP	211335	460
OPT.0.0	65505	225
Gain (%)	69	

TAB. 5 – *Retards au décollage générés par les différentes stratégies*

	Temps d'activité cumulé des positions de contrôle (en minutes)						
	Aix	Bord.	Brest	Paris(E)	Paris(O)	Reims	**Total**
DEP	19230	14160	14700	9600	9150	10500	77340
OPT.0.0	18240	11400	8520	7020	7620	9120	61920
Gain (%)	5.1	19.5	42	26.9	16.7	13.1	19.9

TAB. 6 – *Ressources nécéssaires selon la stratégie choisie*

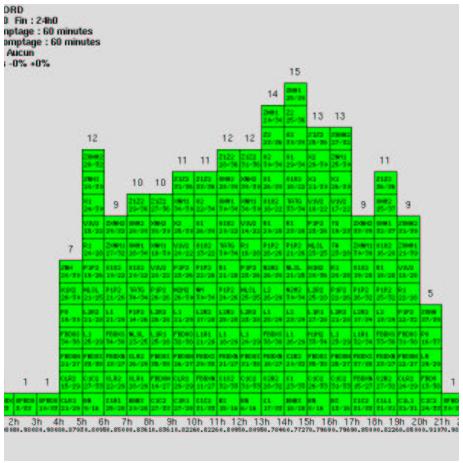

FIG. 3 – *Configurations optimales pour Bordeaux, avec lissage préalable du trafic (OPT.0.0)*

La figure 3 présente le résultat de la stratégie OPT.0.0 pour Bordeaux, avec des tolérances nulles sur les capacités : nous constatons qu'il ne reste plus aucun dépassement de capacité. Sur la figure 4, il est possible de comparer pour chaque moment de la journée le schéma optimisé et le schéma déposé.

9 Conclusion

Les algorithmes classiques et l'algorithme génétique trouvent des configurations optimales de secteurs, en prenant en compte les mêmes paramètres et contraintes que dans le processus FMP/CFMU. Les résultats sont obtenus en un temps compatible avec une utilisation

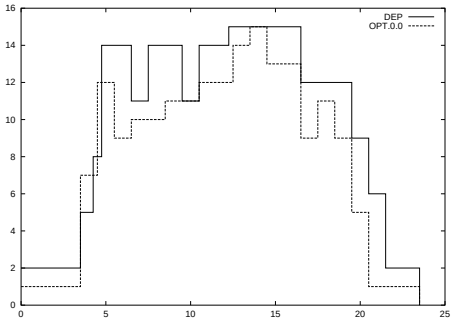

FIG. 4 – *Comparaison entre OPT.0.0 et DEP pour Bordeaux*

opérationnelle. Les algorithmes de recherche dans des arbres sont plus rapides que l'algorithme génétique, pour les centres français et dans le contexte des groupes de secteurs issus des données des centres.

Cependant, la relation combinatoire entre le nombre de secteurs et le nombre de partitions et de configurations est telle que les algorithmes classiques risquent d'être impraticables dans le contexte de plus grands centres de contrôle ou si l'on souhaite élargir l'ensemble des groupes de secteurs opérationnellement utilisables. Dans un tel cas, l'algorithme génétique est une bonne alternative. Il fournit plusieurs configurations optimales ou presque optimales en un temps de calcul borné par l'utilisateur.

Les résultats sur les stratégies d'allocation de créneaux sont bons : environ 70% de gain en délais en utilisant environ 20% moins de ressources. Cependant, la nouvelle stratégie ne tient pas compte des incertitudes sur la prévision de trafic. De plus, les flux de trafic et les capacités horaires sont de mauvais indicateurs de la charge de travail du contrôleur aérien et de son seuil de saturation. En conséquence, il n'est pas certain qu'une méthode optimisant les délais alloués et les schémas d'ouvertures sur la base de ces indicateurs conduise vraiment à un réel lissage de la charge de travail des contrôleurs. Notons que ces remarques sont également valables pour la méthode actuelle (DEP), mais celle-ci a au moins l'avantage d'être validée par un retour d'expérience.

Jusqu'à présent, nous avons considéré l'optimisation des configurations de secteurs uniquement dans le contexte des schémas d'ouvertures prévisionnels réalisés par les FMP, sur la base des flux de trafic prévus et des capacités horaires. Les algorithmes proposés pourraient également être envisagés, avec une définition adaptée de la charge de travail et des seuils et contraintes, pour proposer en temps réel des configurations de secteurs aux chefs de salles de contrôle.

Références

[BBR01] N. Barnier, P. Brisset, and T. Rivière. Slot allocation with constraint programming: Models and results. In *Proceedings of the fourth USA/Europe Air Traffic Management R&D Seminar*, 2001.

[DA98] N. Durand and J. M. Alliot. Genetic crossover operator for partially separable functions. In *Proceedings of the third annual Genetic Programming Conference*, 1998.

[DASF95] Daniel Delahaye, Jean-Marc Alliot, Marc Schoenauer, and Jean-Loup Farges. Genetic algorithms for automatic regroupement of air traffic control sectors. In *Proceedings of the Conference on Evolutionary Programming*, 1995.

[Gol89] David Goldberg. *Genetic Algorithms*. Addison Wesley, 1989. ISBN: 0-201-15767-5.

[Mic92] Zbigniew Michalewicz. *Genetic algorithms+data structures=evolution programs*. Springer-Verlag, 1992. ISBN: 0-387-55387-.

[Pea84] Judea Pearl. *Heuristics*. Addison-Wesley, 1984. ISBN: 0-201-05594-5.

[VM01] C. Verlhac and S. Manchon. Optimization of opening schemes. In *Proceedings of the fourth USA/Europe Air Traffic Management R&D Seminar*, 2001.