Optimisation combinatoire en temps limité : Depth First Branch and Bound adaptatif

Simon de Givry — Pierre Savéant — Jean Jourdan

Thomson-CSF LCR, laboratoire A&TS
Domaine de Corbeville
91404 Orsay cedex
{degivry, saveant, jourdan}@lcr.thomson-csf.com

RÉSUMÉ : *Pour des problèmes d'optimisation combinatoire, nous proposons une approche visant à prendre en compte une durée imposée pour la recherche des solutions. Nous introduisons dans l'algorithme Depth First Branch and Bound des paramètres pour régler sa complexité. Ces paramètres fixent le compromis entre rapidité et complétude. Nous étudions diverses stratégies pour régler dynamiquement ces paramètres en fonction du domaine d'application, de l'instance à résoudre et du contrat de temps imparti. Enfin, nous montrons l'efficacité de notre approche sur une application de planification d'un système d'arme.*

MOTS-CLÉS : *Optimisation combinatoire, Depth First Branch and Bound, contrat temporel, Iterative Broadening*

1. Introduction

Les *problèmes d'optimisation combinatoire* apparaissent dans les situations où un certain nombre de choix parmi un ensemble fini d'alternatives doivent être décidés en vue d'obtenir une solution optimale. Le critère de coût associé fournit une mesure quantitative de la qualité de la solution. La plupart des problèmes d'optimisation combinatoire étant NP-difficiles, l'intérêt s'est porté sur des méthodes de résolution approchées qui permettent d'obtenir de bonnes solutions dans un temps raisonnable. Ces méthodes se scindent en deux grandes familles : les méthodes constructives et les méthodes de recherche locale. Nous nous restreindrons à la première famille dans la suite de l'article. Les méthodes constructives se traduisent habituellement par une recherche arborescente incomplète en profondeur d'abord en s'appuyant sur des heuristiques [PEA 84, LAN 92, HAR 95a, BRE 96, WAL 97].

Le contexte opérationnel, par ailleurs, peut imposer un certain laps de temps à ne pas dépasser pour traiter le problème. C'est le cas notamment des systèmes en ligne tels que les systèmes temps réel qui imposent le respect d'échéances temporelles, ainsi que les systèmes interactifs pour lesquels une durée d'attente maximum pour fournir une solution à l'utilisateur est fixée implicitement.

Dans ce contexte, l'algorithme mis en œuvre doit non seulement respecter le contrat de temps mais aussi l'exploiter le mieux possible.

La première question nous renvoie au problème de l'estimation du temps de résolution d'un algorithme non-déterministe, sujet abordé notamment par [KNU 75, PUR 78, CHE 92] pour des problèmes de satisfaction, et repris par [LOB 98b] pour des problèmes d'optimisation.

Diverses approches ont été proposées pour répondre à la deuxième question. Dans [CHU 91, CHU 94], l'algorithme $A^\star_\epsilon$ [PEA 84] est adapté en fonction du contrat de temps en jouant sur la vitesse de convergence du majorant. Les stratégies itératives proposés dans [GIN 92, HAR 95b, WAL 97] consistent à enchaîner le même algorithme en «ouvrant» progressivement les points de choix. L'approche itérative convient particulièrement au mode interruptible mais présente l'inconvénient de répéter des étapes identiques pour le mode contractuel. Dans [BRE 96], la stratégie consiste à répéter un grand nombre de fois un algorithme glouton dont l'heuristique est biaisée de façon aléatoire. Dans [ALL 96] (resp. [LOB 98a]), pour des problèmes de satisfaction (resp. optimisation), l'algorithme complet estimé comme le plus rapide est selectionné parmi une liste d'algorithmes potentiels en fonction de l'instance à résoudre.

Dans cet article nous proposons un schéma d'adaptation dynamique de la complexité de l'algorithme fonction du domaine d'application, de l'instance à résoudre et du contrat de temps imparti.

Après avoir identifié les paramètres réglant la complétude de l'algorithme, nous appliquons une stratégie d'adaptation dynamique de ces paramètres en fonction du temps restant et du «chemin» déjà parcouru.

Selon l'inventaire dressé dans [LOB 97], notre approche s'inscrit dans un schéma de coopération calculée entre méthodes systématiques et approchées. Dans ce schéma le compromis entre rapidité et complétude est tranché en cours de résolution.

Un système en ligne qui impose une limite sur le temps de calcul et exige l'obtention d'une solution complète à la fin de ce temps, ne peut traiter que des problèmes d'optimisation où il est possible de trouver une première solution, même de qualité médiocre, dans un laps de temps inférieur au contrat. Nous nous plaçons sous cette hypothèse de contrat minimum.

Dans la section 2 nous analysons le paramétrage de la complexité d'un algorithme *Depth First Branch and Bound* incomplet. Un nouvel algorithme *Depth First Branch and Bound* paramétré y est également présenté. Trois stratégies de réglage des paramètres, éventuellement combinables entre elles, sont proposées dans la section 3. Le schéma précédent est enrichi pour aboutir à un algorithme *Depth First Branch and Bound* adaptatif. Enfin dans la section 4, nous comparons divers algorithmes paramétrés sur une application de planification d'un système d'arme.

2. Algorithme Depth First Branch and Bound incomplet paramétré

Les méthodes constructives s'appuient sur des heuristiques (générales ou spécifiques au domaine d'application) pour guider l'ordre des choix de la recherche. C'est à partir de ces heuristiques qu'il est possible de construire des algorithmes *Depth First Branch and Bound* incomplets offrant de meilleures performances en temps limité qu'un simple *Depth First Branch and Bound* complet.

Les algorithmes seront décrits à l'aide d'un pseudo-langage procédural. Pour des raisons de clarté tous les algorithmes utilisent un ordre statique sur les variables. Sans perte de généralité, nous nous restreignons au cadre des problèmes de *minimisation*.

2.1. *Algorithme Depth First Branch and Bound*

La figure 1 présente l'algorithme *Depth First Branch and Bound* standard [1] avec un ordre dynamique sur les valeurs.

```
DFBB()
    ub ← +∞
    SEARCH-DFBB(1)
SEARCH-DFBB(i)
    D ← SORTED-DOMAIN(i, H)
    for k = 1 to DS(i)
        A[i] ← D[k]
        lb ← LB(A, i)
        if lb < ub then if i = n then ub ← lb
                                  else SEARCH-DFBB(i + 1)
```

Figure 1. *Depth First Branch and Bound*

La variable globale ub reçoit le majorant courant de l'optimum du problème. Le problème comporte n variables et DS(i) retourne la taille du domaine de la variable i. Un tableau noté A contient l'affectation courante des variables. La fonction LB(A, i) retourne un minorant du sous-problème caractérisé par l'affectation des variables de 1 à i contenue dans A. Lorsque toutes les variables sont instanciées, LB(A, n) retourne la valeur du critère à optimiser.

Nous considérons qu'une heuristique sur les valeurs fournit une mesure (valeur entière ou réelle) de la qualité d'un choix d'une valeur vis à vis du critère à optimiser. $H(A, i, k)$ mesure l'intérêt d'affecter la valeur k à la variable i, connaissant l'affectation des variables de 1 à $i - 1$. La fonction SORTED-DOMAIN(i, H) retourne le domaine de la variable i trié suivant l'heuristique H.

1. La variante qui consiste à repartir de la racine avec le dernier majorant trouvé n'est pas traitée dans cet article.

2.2. *Paramétrage de la complexité de l'algorithme*

Le paramétrage doit, à partir d'un même schéma, permettre l'obtention d'algorithmes incomplets, intermédiaires entre un algorithme glouton et un algorithme complet. Une première étape consiste alors à exhiber des paramètres qui limitent l'exploration.

L'incomplétude peut intervenir à plusieurs endroits dans l'algorithme DFBB. Nous suggérons trois voies pour limiter une exploration :

— limiter l'énumération des valeurs d'une variable (fonction SORTED-DOMAIN modifiée) ;

— limiter l'arborescence développée pour un sous-ensemble de variables (appel récursif à DFBB modifié) ;

— limiter les valeurs que peut prendre le majorant ub (test $lb < ub$ modifié).

Dans les deux premiers cas, le principe consiste à interdire des valeurs ou des instanciations de plusieurs variables qui sont mal notées par les heuristiques. Ceci revient à faire l'hypothèse que les bonnes solutions sont composées de valeurs bien notées par les heuristiques. Néanmoins les heuristiques ne sont pas parfaites. Par exemple, elles sont généralement moins informées en haut de l'arbre de recherche qu'en bas [WAL 97].

2.2.1. *Paramètres pour limiter l'énumération des valeurs d'une variable*

— nombre maximum des meilleures valeurs retenues ;

— distance maximum, en terme de mesure de l'heuristique, à la meilleure valeur du domaine en dessous de laquelle une valeur est retenue ; de façon à rendre indépendant ce paramètre de la variable, la distance maximum peut s'exprimer en pourcentage (ratio entre la distance à la meilleure valeur et, par exemple, la plus grande distance observée sur un jeu d'instances ou dynamiquement en cours d'exploration) ;

— degré de regroupement des valeurs en classes de valeurs similaires ; au plus une valeur de chaque classe est retenue ; une classe est soit définie par la note donnée par l'heuristique, soit par un comportement similaire des valeurs vis à vis des contraintes du problème [FRE 91, FRE 95, GIV 98].

Ces paramètres sont propres à chaque variable, ou bien à chaque type de variable suivant le domaine d'application, ou encore fonction de la profondeur dans l'arbre de recherche.

2.2.2. *Paramètres pour limiter l'arborescence développée pour un sous-ensemble de variables*

— conditions topologiques sur l'arborescence indépendantes des heuristiques : nombre maximum de nœuds, nombre maximum de backtrackings, nombre maximum de feuilles du sous-arbre ;

— nombre maximum des meilleures feuilles du sous-arbre, retenues par une recherche en avant (*look-ahead*), éventuellement elle aussi incomplète ; également, distance maximum (en pourcentage) à la meilleure feuille du sous-arbre en dessous de laquelle une feuille est retenue ;

— nombre maximum d'erreurs commises par rapport à l'heuristique ; dans une branche, le nombre d'erreurs peut être calculé comme étant la somme des indices moins un, dans les domaines triés par l'heuristique, des valeurs affectées aux variables (extension à l'optimisation et sur des domaines de taille supérieure à deux du paramètre *discrepancy* [HAR 95b]).

Ce type de coupe peut s'appliquer à des sous-ensembles de variables déterminés en fonction du domaine d'application, ou à l'ensemble des variables dans le cas du nombre maximum d'erreurs à l'heuristique. Là encore, nous pouvons exhiber un nouveau paramètre réglant la taille d'un sous-ensemble de variables (par exemple, la profondeur du *look-ahead*).

Plusieurs heuristiques sur les valeurs peuvent s'avérer pertinentes pour le problème à résoudre. Soit, il est possible de les combiner pour n'en faire qu'une : soit, plusieurs domaines triés du même domaine initial de valeurs sont à considérer. Les paramètres vus précédemment peuvent alors être définis sur chacun des domaines triés. De nouveaux paramètres contrôlent la combinaison des choix dans ces domaines triés, comme le nombre maximum de domaines considérés, la distance maximale moyenne en pourcentage d'une valeur par rapport aux meilleures valeurs des différents domaines triés, etc ...

Dans cet article, nous nous focalisons essentiellement sur des heuristiques concernant les valeurs. Cependant, dans le cas d'algorithmes *Depth First Branch and Bound* incomplets, les heuristiques sur les variables jouent également un rôle important pour sélectionner des zones prometteuses dans l'espace des solutions. En effet, deux parcours gloutons avec la même heuristique sur les valeurs et deux heuristiques sur les variables différentes ne donnent pas forcément la même solution. Une heuristique sur les valeurs peut être plus ou moins bien informée suivant les variables précédemment instanciées. Il serait intéressant d'étudier des paramètres réglant le nombre d'heuristiques sur les variables considérées ainsi qu'une modification de l'algorithme DFBB visant à intégrer plusieurs heuristiques sur les variables (examen de plusieurs variables à un même niveau de profondeur)[2].

2.2.3. *Paramètre pour accélérer la convergence du majorant*

La vitesse de convergence de ub vers l'optimum, notée ϵ ($0 < \epsilon \leq 1$), vient s'ajouter dans le test de coupe du *Branch and Bound* : $lb < \epsilon \times ub$ [PEA 84]. Ce paramètre limite les valeurs que peut prendre le majorant ub, en faisant l'hypothèse qu'à chaque nouvelle solution trouvée, il existe des solutions de coût inférieur à $\epsilon \times ub$[3].

---

2. Cette modification augmente a priori la complexité de l'algorithme DFBB complet. Elle n'a de sens que si le parcours en dessous de chaque variable est très limité.

3. Dans le cas où $\epsilon \times ub$ est inférieur ou égal à l'optimum, l'algorithme ne produit plus de solution mais seulement un minorant de l'optimum lorsqu'il termine (égal à $\epsilon \times ub$). Dans ce cas l'algorithme garantit que la meilleure solution rendue est à une distance relative de $1 - \epsilon$ de l'optimum.

2.3. *Exemple d'un algorithme Depth First Branch and Bound incomplet paramétré*

L'exemple qui suit (cf. Figure 2) est un algorithme *Depth First Branch and Bound* paramétré sur le nombre maximum des meilleures valeurs retenues (α) et sur la distance maximum à la meilleure valeur du domaine en dessous de laquelle une valeur est retenue (δ).

```
DFBB-PARAM()
   ub ← +∞
   SEARCH-DFBB-PARAM(1)
SEARCH-DFBB-PARAM(i)
   D ← SORTED-DOMAIN(i, H)
   if UPDATE-δH(i, D) then UPDATE-L(1, i) else UPDATE-L(i, i)
   k ← 1
   while k ≤ L[i]
         A[i] ← D[k]
         lb ← LB(A, i)
         if lb < ub then if i = n then ub ← lb
                                  else SEARCH-DFBB-PARAM(i + 1)
         k ← k + 1
UPDATE-δH(i, D)
   updated ← false
   for k = 1 to DS(i)
      δH[i, k] ← H(A, i, D[k]) − H(A, i, D[1])
      if δH[i, k] > δHmax then δHmax ← δH[i, k]
                               updated ← true
   return updated
UPDATE-L(s, e)
   for i = s to e
      k ← min(α, DS(i))
      while k ≥ 2 ∧ δH[i, k] > δ × δHmax
            k ← k − 1
      L[i] ← k
```

Figure 2. *Depth First Branch and Bound paramétré*

Le tableau $\delta H[i, k]$ contient la distance suivant H entre la $k$ième valeur et la première valeur du domaine trié de la variable i. $L[i]$ contient la limite courante sur le nombre de valeurs retenues du domaine trié de la variable i.

La limite en pourcentage δ implique une limite concrète égale à $\delta \times \delta Hmax$ avec $\delta Hmax$, la plus grande valeur du tableau δH. $\delta Hmax$ est mis à jour dynamiquement en cours d'exploration.

La fonction UPDATE-$\delta H(i, D)$ met à jour le tableau δH pour la variable courante

i. Elle retourne la valeur $true$ si $\delta Hmax$ a été modifié. La fonction UPDATE-$L(s, e)$ met à jour le tableau L de la variable s à la variable e.

3. Réglage des paramètres de complétude

La stratégie est fondée sur le constat suivant, observé empiriquement :

« pour une instance donnée, plus le temps de résolution est proche du contrat de temps, plus la qualité de la solution a des chances d'être meilleure ».

Avec un réglage qui produit un temps de résolution trop court, l'algorithme s'interdit des parties de l'espace de recherche qu'il aurait eu le temps de parcourir. Un réglage trop long empêche l'algorithme d'aller jusqu'au bout. [4]

Le but est donc de régler l'algorithme de telle sorte à utiliser au plus le temps disponible sans le dépasser si possible.

3.1. *Réglage a priori*

Les paramètres sont fixés à partir d'expérimentations sur un jeu d'instances caractéristiques du domaine d'application traité.

Par exemple, l'algorithme sera glouton sur certaines variables, alors que pour d'autres il effectuera une recherche limitée en largeur par un nombre maximum de valeurs par variable ou de feuilles. Cette limite est fixée pour un contrat de temps donné d'après les expérimentations.

Pour guider ce réglage empirique, il est intéressant d'observer l'efficacité de l'heuristique sur les valeurs en fonction du type de variable ou de la profondeur dans l'arbre de recherche. Les meilleures solutions trouvées pour les différentes instances (hors contrat de temps) sont mémorisées en notant pour chaque solution, les indices des valeurs affectées aux variables dans les domaines triés. Pour les indices proches de un en moyenne, l'heuristique s'avère pertinente et l'algorithme peut être glouton sur les variables correspondantes.

Notons que le réglage a priori ne prend en compte que le domaine d'application et le contrat de temps mais pas l'instance.

3.2. *Réglage progressif*

Le réglage progressif consiste à enchaîner une succession d'exécutions d'un même algorithme constructif en faisant croître la valeur des paramètres de complétude jus-

---

4. De la même façon qu'un algorithme *Depth First Branch and Bound* complet interrompu produit généralement de moins bonnes solutions qu'un algorithme *Depth First Branch and Bound* incomplet qui termine. En effet, l'algorithme complet va passer du temps en bas de l'arbre à s'éloigner de l'heuristique avant de tenter des éloignements à l'heuristique en haut de l'arbre qui aurait pu être profitables. C'est également ce que l'algorithme LDS propose d'éviter.

qu'à la fin du contrat.

Cette stratégie itérative, dite interruptible, permet de s'adapter à différentes instances et également à des contrats de temps variables.

La contre-partie est qu'en général même si elle profite d'un meilleur majorant trouvé aux étapes précédentes, l'étape n reparcours les nœuds visités à l'étape $n-1$.

L'algorithme Iterative Broadening [GIN 92], voir la figure 3, reprend l'algorithme DFBB en y introduisant une limite, notée α, sur le nombre des meilleures valeurs retenues par domaine. Sa stratégie itérative consiste simplement à enchaîner l'algorithme SEARCH-DFBB-IB avec une valeur α croissant de un en un jusqu'à la taille maximum du plus grand domaine. La première étape est une descente gloutonne. La dernière étape est une recherche complète, identique à l'algorithme DFBB.

```
IB()
    ub ← +∞
    for d = 1 to max_{i∈[1,n]}(DS(i))
        α ← d
        SEARCH-DFBB-IB(1)
SEARCH-DFBB-IB(i)
    D ← SORTED-DOMAIN(i, H)
    for k = 1 to min(α, DS(i))
        A[i] ← D[k]
        lb ← LB(A, i)
        if lb < ub then if i = n then ub ← lb
                                 else SEARCH-DFBB-IB(i + 1)
```

Figure 3. *Iterative Broadening [GIN 92]*

Notons que le réglage progressif est indépendant du domaine d'application, de l'instance et du contrat. Cependant, un algorithme itératif s'adapte de manière plus ou moins grossière au contrat de temps et à l'instance dans le sens où l'avant dernière étape correspond à la plus grosse recherche arborescente non interrompue.

3.3. *Réglage adaptatif*

Alors que les réglages précédents étaient indépendants de l'instance ou du contrat de temps, le réglage adaptatif cherche le bon réglage en fonction du contrat de temps et de l'instance. L'objectif du réglage adaptatif est d'effectuer une unique recherche incomplète qui termine proche de la fin du contrat de temps. Or il n'existe pas a priori de fonction mathématique qui prenne en entrée un contrat et retourne un jeu de valeurs associées aux paramètres. En effet, le problème inverse, c'est-à-dire l'estimation du temps pris par la recherche en fonction de la valeur des paramètres, n'est traitable que de façon empirique de manière à s'adapter à l'instance. De plus, plusieurs réglages des paramètres peuvent conduire à la même estimation du temps. Il faut donc imposer une stratégie de choix parmi les réglages. Cette stratégie doit favoriser l'ouverture de la recherche dans des zones de moins en moins préférées par l'heuristique plus le contrat de temps est grand.

Dans la suite nous faisons l'hypothèse d'un temps moyen pour développer un nœud ce qui sous-entend que le calcul des minorants ($\text{LB}(A, i), \forall A, \forall i \in [1, n]$) ne fait pas appel à un algorithme trop complexe et donc que leurs temps de calcul varient peu.

3.3.1. *Estimation du temps de résolution en fonction des paramètres de complétude*

Nous avons identifié trois méthodes empiriques pour estimer le temps d'une résolution à partir d'un jeu de paramètres donné :

— estimation de la taille de l'arbre de recherche par une série de sondages réalisés en pré-traitement (méthode [KNU 75, PUR 78, CHE 92, LOB 98b]) ; à partir de cette estimation ainsi que du temps moyen pour développer un nœud, nous en déduisons une estimation du temps de parcours de l'arbre ;

— durant la recherche, acquisition de la profondeur moyenne (p), ainsi que du nombre moyen de valeurs retenues par variable (d) ; puis calcul de la taille d'un arbre de hauteur p et de facteur de branchement d ; le produit de cette taille par le temps moyen pour développer un nœud nous donne une estimation du temps de parcours de l'arbre réellement parcouru ; cela fait l'hypothèse que la coupe est uniforme ;

— durant la recherche, calcul du pourcentage de l'espace parcouru par rapport à l'espace total à parcourir ; connaissant le temps déjà consommé, nous en déduisons le temps pour terminer ; cela fait aussi l'hypothèse que la coupe est uniforme.

La première méthode est a priori la plus saine d'un point de vue mathématique : c'est un estimateur non biaisé. Néanmoins nous avons choisi d'expérimenter la dernière méthode pour sa simplicité et son faible temps de calcul (linéaire sur le nombre de variables). De plus, cette méthode reconfigure les paramètres régulièrement. Une mauvaise estimation à un instant donné est éventuellement corrigée par de meilleures estimations aux instants suivants.

3.3.2. *Stratégie de réglage automatique des paramètres*

Comment trouver un jeu de valeurs aux paramètres qui implique un arbre de recherche dont le temps de parcours soit proche de la durée du contrat ? Potentiellement, le nombre de combinaisons devient vite très grand dès qu'il y a plusieurs paramètres à grands domaines de variation. Un examen exhaustif des combinaisons n'est pas réaliste. De plus, différents réglages peuvent donner la même estimation. Or il est possible que parmi ces réglages, il en existe un qui soit préférable car il produit des solutions de meilleure qualité en moyenne sur un ensemble d'instances du domaine d'application.

C'est pourquoi nous proposons d'utiliser cette connaissance du domaine pour établir une liste de jeux de valeurs des paramètres pertinents. Nous définissons une stratégie S comme étant un tableau contenant cette liste.

Pour simplifier le réglage automatique, cette liste doit être ordonnée de telle sorte que :

$$\forall I,\ i < j \quad \Rightarrow$$
$$\text{ESTIMATED-REMAINING-TIME}(I, S[i]) \leq \text{ESTIMATED-REMAINING-TIME}(I, S[j])$$

avec ESTIMATED-REMAINING-TIME$(I, S[i])$, une estimation du temps restant pour que l'algorithme incomplet termine sur l'instance I avec le jeu de valeurs des paramètres $S[i]$.

En pratique, pour que la condition soit vraie quelle que soit l'instance, la liste des jeux de valeurs des paramètres est construite en faisant croître les valeurs de façon monotone.

De cette manière, le meilleur réglage pour une instance, un contrat et une stratégie S donnés, est rapidement trouvé par dichotomie en $O(\log_2 \phi_{max})$ étapes, avec ϕ_{max}, le nombre d'éléments de S. La stratégie permet de capitaliser une expertise sur un domaine d'application.

Pour une estimation calculée en cours d'exploration, nous devons définir quand et sous quelles conditions, le réglage des paramètres peut s'effectuer :

— avant l'exploration, les paramètres sont initialisés avec $S[\phi_{max}]$ pour ne pas terminer prématurément;

— pendant l'exploration, les paramètres sont mis à jour avec une certaine fréquence (σ); par exemple, tous les σ backtrackings; dans le cas d'une utilisation de l'estimation à partir du pourcentage de l'espace parcouru, il est préférable de faire le réglage au niveau des feuilles de manière à avoir un pourcentage plus précis; le réglage peut être interdit près de la racine, au dessus d'une profondeur (ν) décidée a priori ;

— une condition générale pour décider s'il y a réglage consiste à mesurer l'écart relatif entre ESTIMATED-REMAINING-TIME et DEADLINE :

$$\frac{|\text{ESTIMATED-REMAINING-TIME}(I, S[\phi]) - \text{DEADLINE}()|}{\text{DEADLINE}()} > \epsilon$$

avec ϵ, une valeur proche de zéro et DEADLINE, une fonction qui retourne le temps restant avant la fin du contrat.

3.3.3. *Algorithme Depth First Branch and Bound incomplet adaptatif*

L'algorithme DFBB-PARAM est étendu en introduisant un contrôle dynamique des paramètres (fonctions ADJUST-PARAMETERS et MOVE). L'estimation du temps nécessaire pour terminer l'exploration s'appuie sur le pourcentage de l'espace parcouru (fonctions %-OF-SEARCH-DONE et ESTIMATED-REMAINING-TIME). La figure 4 décrit l'algorithme DFBB-ADAPT ainsi obtenu.

La variable globale ϕ indique l'indice du jeu courant $S[\phi]$ de valeurs des paramètres. Le tableau K contient par variable l'indice courant de sa valeur dans son domaine trié.

```
DFBB-ADAPT()
    ub ← +∞
    (α, δ) ← S[φ_max] ; φ ← φ_max ; bt ← 0
    SEARCH-DFBB-ADAPT(1)
SEARCH-DFBB-ADAPT(i)
    leaf ← true
    D ← SORTED-DOMAIN(i, H)
    if UPDATE-δH(i, D) then UPDATE-L(1, i) else UPDATE-L(i, i)
    k ← 1
    while k ≤ L[i]
            K[i] ← k
            A[i] ← D[k]
            lb ← LB(A, i)
            if lb < ub then if i = n then ub ← lb
                                    else SEARCH-DFBB-ADAPT(i + 1) ; leaf ← false
            k ← k + 1
    if leaf ADJUST-PARAMETERS(i)
%-OF-SEARCH-DONE(e)
    s ← 0 ; p ← 1
    for i = 1 to e
        p ← p × L[i] ; s ← s + (min(K[i], L[i]) − 1) / p
    return s + 1/p
ESTIMATED-REMAINING-TIME(i)
    r ← %-OF-SEARCH-DONE(i)
    return (ELAPSED-TIME() × (1 − r)) / r
ADJUST-PARAMETERS(i)
    bt ← bt + 1
    if bt mod σ = 0 ∧ i ≥ ν
    then τ ← ESTIMATED-REMAINING-TIME(i)
         Δ ← DEADLINE()
         if |τ − Δ| / Δ > ε then if τ < Δ then MOVE(i, Δ, φ, τ, φ_max, +∞)
                                         else MOVE(i, Δ, 1, 0, φ, τ)
MOVE(i, Δ, φ_inf, τ_inf, φ_sup, τ_sup)
    if φ_sup − φ_inf ≥ 2 then φ_med ← (φ_sup + φ_inf) / 2
                              (α, δ) ← S[φ_med] ; UPDATE-L(1, i)
                              τ_med ← ESTIMATED-REMAINING-TIME(i)
                              if τ_med < Δ then MOVE(i, Δ, φ_med, τ_med, φ_sup, τ_sup)
                                           else MOVE(i, Δ, φ_inf, τ_inf, φ_med, τ_med)
                         else (α, δ) ← S[φ_sup] ; UPDATE-L(1, i)
                              φ ← φ_sup
```

Figure 4. *Depth First Branch and Bound adaptatif*

4. Expérimentations sur une application de planification d'un système d'arme

L'application qui a servi de banc d'essais concerne la chaîne d'engagement d'un système d'arme. Il s'agit de fournir de façon cyclique un plan d'engagement à partir des pistes [5] atteignables et des missiles disponibles. Un plan d'engagement correspond à un ensemble d'associations piste-missile avec une date de tir.

L'activation d'un missile se fait en plusieurs phases et nécessite l'emploi d'un certain nombre de ressources. A chaque piste est associé un segment d'atteignabilité qui est un intervalle temporel pendant lequel elle est atteignable. Un missile peut être associé à une piste si sa date de tir au plus tôt appartient au segment d'atteignabilité de celle-ci. Le critère d'optimisation vise entre autres choses à minimiser le nombre de pistes pour lesquelles un missile n'a pu être affecté.

Après une modélisation en contraintes, diverses expérimentations ont été menées dans le but d'étudier les performances des algorithmes décrits dans les sections précédentes.

Le réglage des algorithmes paramétrés pour cette application se situe au niveau du choix d'un missile pour une piste et du choix d'une ressource pour un missile.

Les paramètres δ_{miss} et δ_{ress} expriment une distance maximum à la meilleure valeur donnée par l'heuristique et α_{miss} dénote le nombre maximum de meilleures valeurs retenues, données par l'heuristique.

Une première comparaison a porté sur un réglage dynamique par rapport à un réglage statique. Les trois algorithmes suivants ont été mis en œuvre.

— DFBB-PARAM1 avec $\delta_{ress} = 100\%, \delta_{miss} = 100\%, \alpha_{miss} = 1$;

— DFBB-PARAM2 avec $\delta_{ress} = 100\%, \delta_{miss} = 100\%, \alpha_{miss} = 2$;

— DFBB-ADAPT.

La figure 5 montre la valeur du critère à minimiser, en fonction du contrat de temps. Chaque point correspond à une moyenne sur deux mille cycles de la valeur du critère de la meilleure solution trouvée par l'exécution d'un algorithme particulier avec un contrat de temps donné.

DFBB-PARAM1 a un réglage des paramètres tel qu'il termine en au plus 400 millisecondes sur tous les cycles. Il ne profite donc pas d'un contrat de temps supérieur à 400 millisecondes. Par contre DFBB-PARAM2 ne termine pas avant une seconde dans 10% des cas et avant deux secondes dans 4% des cas. Enfin DFBB-ADAPT domine les deux autres algorithmes à partir d'un contrat de 600 millisecondes.

La figure 6 visualise trois aspects du réglage adaptatif de DFBB-ADAPT, en fonction du temps, pour une unique recherche avec un contrat de temps égal à 1 seconde. Après chaque réglage (un appel à la fonction ADJUST-PARAMETERS suivi d'une série d'appels à la fonction MOVE), nous enregistrons la valeur retournée par la fonction %-OF-SEARCH-DONE, la valeur de l'indice ϕ du jeu $S[\phi]$ de valeurs des paramètres (indice normé variant de 0 à 100) et la valeur de l'écart relatif entre

5. Une piste est la représentation mathématique d'une ou plusieurs cibles détectées par un radar.

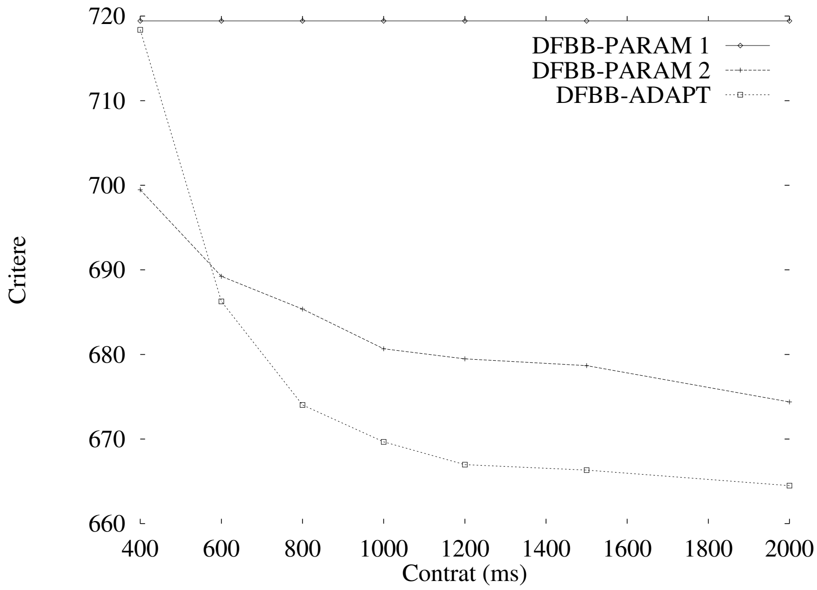

Figure 5. *Comparaison : réglage dynamique versus réglage statique*

ESTIMATED-REMAINING-TIME et DEADLINE (cf. section 3.3.2).

La progression quasi-linéaire de la courbe %-OF-SEARCH-DONE indique une coupe relativement uniforme, sur cet exemple. Pendant les premières 200 millisecondes, l'espace parcouru est encore trop faible pour que l'estimation du temps restant soit bonne. Ensuite, le réglage reste stable jusqu'à une adaptation, à l'approche de la fin du contrat. La troisième courbe montre la justesse du réglage, c-à-d l'écart entre l'estimation du temps restant pour terminer la recherche et le temps restant avant l'échéance. L'écart moyen est seulement de 12% sur cet exemple.

Une seconde campagne de tests a permis une comparaison globale entre un algorithme glouton qui suit les heuristiques à la lettre ($Glouton$), un algorithme itératif avec pour seul paramètre la variable α_{miss} (IB) et un algorithme adaptatif (DFBB-ADAPT). Construite de la même façon que la figure 5, la figure 7 présente les résultats obtenus. Une droite représentant une moyenne des minorants des optima a été ajoutée à la figure 7. Ces minorants ont été obtenus par l'exécution d'un algorithme *Depth First Branch and Bound* complet sur une simplification du problème. Cette approche par simplification [GIV 98] nous permet d'encadrer dans un temps raisonnable (en moyenne une exécution dure une minute) les optima des deux mille cycles du banc d'essais.

Nous pouvons noter que les algorithmes itératifs et adaptatifs améliorent la qualité des solutions en fonction du contrat de temps, ce dont est incapable l'algorithme glouton. Le gain par rapport à l'algorithme glouton est déjà significatif au bout de

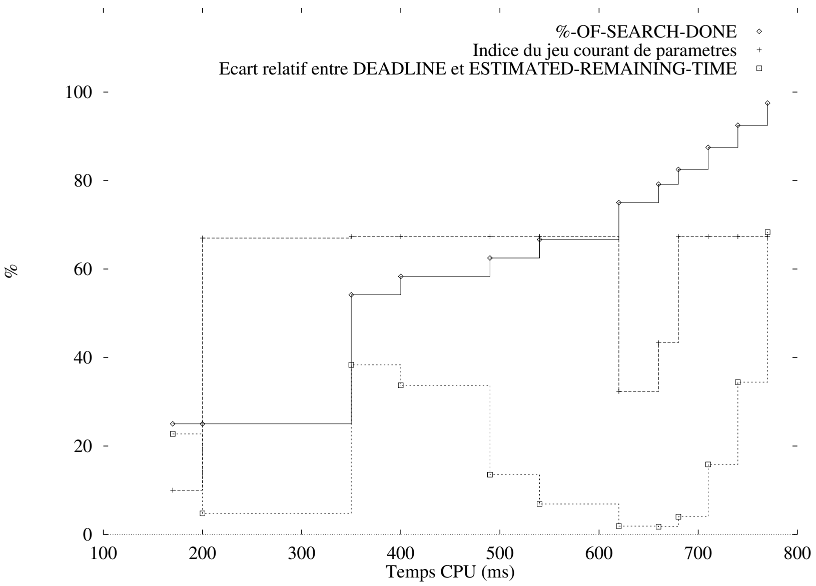

Figure 6. *Examen du réglage dynamique effectué par* DFBB-ADAPT *pour une instance donnée et un contrat égal à 1000 ms.*

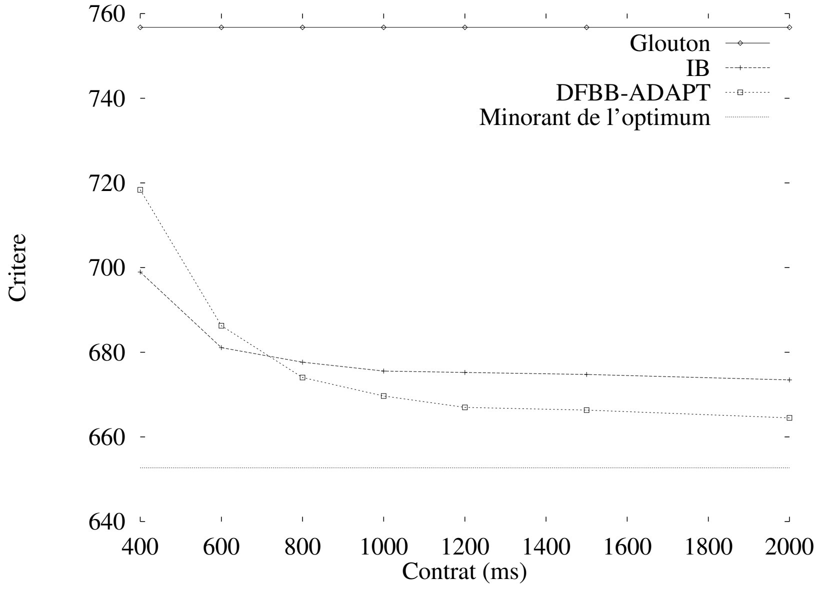

Figure 7. *Comparaison globale : glouton, itératif, adaptatif*

seulement 400 millisecondes.

A noter également que faire évoluer les trois paramètres (δ_{ress}, δ_{miss} et α_{miss}) progressivement serait plus délicat avec une stratégie itérative. L'algorithme adaptatif est plus performant que l'algorithme itératif à partir d'un contrat de temps de 800 millisecondes. Pour des contrats inférieurs, l'estimation du temps d'une résolution est trop imprécis.

5. Conclusion et perspectives

Finalement, une approche pragmatique face à la problématique de l'optimisation combinatoire en temps limité consiste d'abord à mettre au point un réglage a priori en fonction du domaine d'application par une phase d'apprentissage sur un jeu de tests et en le validant à l'aide d'une stratégie itérative. Le réglage a priori permet de déterminer les bons paramètres de complétude et d'en fixer certains. Ensuite, nous pouvons appliquer une stratégie adaptative sur les paramètres restés libres. Ainsi l'algorithme résultant tient compte du domaine d'application, du contrat de temps et de l'instance.

Nous n'avons considéré que des parcours en profondeur d'abord pour éviter des problèmes de mémorisation et de restauration rapide de nœuds liés à des parcours en largeur. Par contre, un *look-ahead* paramétré est envisageable. Il permettrait de simuler une exploration incomplète de type meilleur en premier et viendrait renforcer les heuristiques [FRO 95]. Pour des problèmes de planification temps réel, [KOR 90, HAM 95, PEM 95] utilisent un *look-ahead* paramétré en fonction du contrat de temps pour effectuer des explorations à horizon limité.

De la même façon que nous avons cherché à régler dynamiquement les paramètres d'un algorithme de recherche arborescente, nous pourrions envisager un réglage dynamique d'algorithmes de recherche locale, à l'instar de [ALL 96] pour l'algorithme de recherche tabou et [CHA 99] pour l'algorithme du recuit simulé. Nous serions alors en mesure de proposer des algorithmes hybrides (recherche constructive et recherche locale) adaptatifs.

Bibliographie

[ALL 96] ALLEN J. et MINTON S., « Selecting the Right Heuristic Algorithm: Runtime Performance Predictors ». In *Proceedings of the Canadian AI Conference*, 1996.

[BRE 96] BRESINA J. L., « Heuristic-Biased Stochastic Sampling ». In *Proc. of AAAI-96*, p. 271–278, Portland, OR, 1996.

[CHA 99] CHARON I. et HUDRY O., « Réglage totalement automatique du bruitage ». In *Proc. of ROADEF-99*, p. 120–121, Autrans, France, 1999.

[CHE 92] CHEN P., « Heuristic Sampling : a Method for Predicting the Performance of Tree Searching Programs ». *SIAM Journal on Computing*, vol. 21, n° 2, p. 295–315, 1992.

[CHU 91] CHU L.-C. et WAH B. W., « Optimization in Real Time ». In *Proc. of the Twelfth Real Time Systems Symposium*, p. 150–159, Washington, D.C., 1991.

[CHU 94] CHU L.-C., « *Algorithms for Combinatorial Optimization in Real Time and their Automated Refinements by Genetics-Based Learning* ». PhD thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, IL, 1994.

[FRE 91] FREUDER E. C., « Eliminating Interchangeable Values In Constraint Satisfaction Problems ». In *Proc. of AAAI-91*, p. 227–233, Anaheim, CA, 1991.

[FRE 95] FREUDER E. C. et SABIN D., « Interchangeability Supports Abstraction and Reformulation for Constraint Satisfaction ». In *Proc. of the Symposium on Abstraction, Reformulation, and Approximation (SARA-95)*, p. 62–68, 1995.

[FRO 95] FROST D. et DECHTER R., « Look-ahead value ordering for constraint satisfaction problems ». In *Proc. of IJCAI-95*, p. 572–578, Montréal, Canada, 1995.

[GIN 92] GINSBERG M. et HARVEY W., « Iterative Broadening ». *Artificial Intelligence*, vol. 55, p. 367–383, 1992.

[GIV 98] DE GIVRY S., « *Algorithmes d'optimisation sous contraintes étudiés dans un cadre temps réel* ». PhD thesis, École Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, France, 1998.

[HAM 95] HAMIDZADEH B. et SHEKHAR S., « Deadline Compliance, Predictability, and Online Optimization in Real-Time Problem Solving ». In *Proc. of IJCAI-95*, p. 220–226, Montréal, Canada, 1995.

[HAR 95a] HARVEY W. D., « *NONSYSTEMATIC BACKTRACKING SEARCH* ». PhD thesis, Stanford University, March 1995.

[HAR 95b] HARVEY W. D. et GINSBERG M. L., « Limited Discrepancy Search ». In *Proc. of IJCAI-95*, p. 607–613, Montréal, Canada, 1995.

[KNU 75] KNUTH D., « Estimating the Efficiency of Backtrack Programs ». *Mathematics of Computation*, vol. 29, n° 129, p. 121–136, 1975.

[KOR 90] KORF R. E., « Real-Time Heuristic Search ». *Artificial Intelligence*, vol. 42, p. 189–211, 1990.

[LAN 92] LANGLEY P., « Systematic and Nonsystematic Search Strategies ». In *Proc First Int. Conf. AI Planning Systems*, p. 145–152, 1992.

[LOB 97] LOBJOIS L. et LEMAÎTRE M., « Coopération entre méthodes complètes et incomplètes pour la résolution de (V)CSP: une tentative d'inventaire ». In *Proc. of JNPC-97*, p. 67–73, Rennes, France, 1997.

[LOB 98a] LOBJOIS L. et LEMAÎTRE M., « Branch and Bound Algorithm Selection by Performance Prediction ». In *Proc. of AAAI-98*, p. 353–358, Madison, WI, 1998.

[LOB 98b] LOBJOIS L. et LEMAÎTRE M., « Adaptation de l'estimateur de Knuth aux problèmes d'optimisation combinatoires ». In *Proc. of RJCIA-98*, p. 171–179, Toulouse, France, 1998.

[PEA 84] PEARL J., *HEURISTICS, Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, 1984.

[PEM 95] PEMBERTON J. C., « k-best: A New Method for Real-Time Decision Making ». In *Proc. of IJCAI-95*, p. 227–233, Montréal, Canada, 1995.

[PUR 78] PURDOM P., « Tree Size by Partial Backtracking ». *SIAM Journal on Computing*, vol. 7, n° 4, p. 481–491, 1978.

[WAL 97] WALSH T., « Depth-bounded Discrepancy Search ». In *Proc. of IJCAI-97*, Nagoya, Japan, 1997.