Quelques variantes de LDS

Nicolas Prcovic
LSIS - Université d'Aix-Marseille III
email: `nicolas.prcovic@univ.u-3mrs.fr`

Résumé

Nous présentons plusieurs variantes de LDS : RLDS, qui diffère de LDS par l'inversion de l'ordre de visite des feuilles de l'arbre à l'intérieur des itérations de l'algorithme, et DWDS, qui limite la somme des profondeurs des divergences au lieu de la somme des divergences. Nous justifions ces variantes par rapport à un modèle probabiliste de répartition des solutions dans l'arbre, conformément aux propriétés standard liées à l'utilisation d'une heuristique dynamique de choix de valeurs. Sachant que les algorithmes basés sur les divergences induisent une génération redondante des nœuds internes de l'arbre, nous présentons les variantes de LDS et de DWDS qui permettent une réduction de cette redondance en échange d'une certaine altération de leur ordre de visite des nœuds afin d'augmenter la souplesse et l'efficacité de ces algorithmes.

1 Introduction

Ces dernières années sont apparues des méthodes de recherche arborescente basées sur le concept de divergence, définie dans le contexte d'une recherche où on utilise une heuristique de choix de successeurs d'un nœud. Une divergence est un choix de successeur qui n'est pas le premier sélectionné par l'heuristique. La caractéristique commune des stratégies de recherche basées sur ce concept de divergence est celle de considérer *dans leur globalité* les divergences pouvant être effectuées pour parvenir à une solution. Ainsi, elles examinent les branches de l'arbre de recherche dans un ordre plus pertinent que la traditionnelle recherche en profondeur d'abord (Depth First Search, DFS). Néanmoins, cette amélioration a un coût : des noeuds internes peuvent être visités plusieurs fois. Le but de cet article est de rationnaliser la conception des méthodes de recherche basées sur les divergences. Nous commençons par rappeler le fonctionnement de la recherche à divergence limitée (LDS). Puis, nous dégageons des critères généraux concernant l'ordre de visite des nœuds de l'arbre de recherche que doivent respecter les stratégies de recherche basées sur les divergences. Ces critères nous permettent, d'une part, de déterminer que LDS ne respecte pas complètement ces critères, et d'autre part, de définir de nouvelles stratégies de recherche (RLDS, DWDS) qui les respectent.

Par ailleurs, comme il existe un problème de redondance dans la visite des nœuds de l'arbre de recherche, nous expliquons comment et dans quelles conditions on peut réduire cette redondance quitte à altérer le bon ordre de visite des nœuds pour obtenir des résultats plus efficaces.

Nous nous plaçons dans le contexte des CSP afin de simplifier nos explications mais il est clair que la plupart de ce qui sera exposé restera valable pour d'autres types de problèmes dès lors qu'ils respecteront la propriété suivante : la solution d'un problème ne peut être atteinte qu'après avoir effectué exactement n choix, n étant une constante connue avant la résolution du problème.

2 La recherche à divergence limitée

Dans cette section, nous introduisons la recherche à divergence limitée (LDS) telle qu'elle a été définie originellement dans [HG95]. Elle repose sur l'existence d'une heuristique de choix de valeur guidant la recherche vers les feuilles les plus prometteuses. Par convention, on considérera que le fils gauche issu d'un nœud correspond à un respect de l'heuristique tandis que les autres fils correspondent à une divergence. Cette méthode a été définie sur les arbres de recherche binaires mais est facilement extensible aux arbres d'arité supérieure.

La recherche à divergence limitée (Limited discrepancy search, LDS) [HG95] examine en premier les branches de l'arbre qui accumulent le plus petit nombre de divergences. LDS se base implicitement sur une propriété naturelle que nous appellerons *h-cohérence* et qui se résume informellement par : quelle que soit la profondeur dans l'arbre, suivre le premier choix préconisé par l'heuristique donne plus de chance d'atteindre une solution que suivre un autre choix.

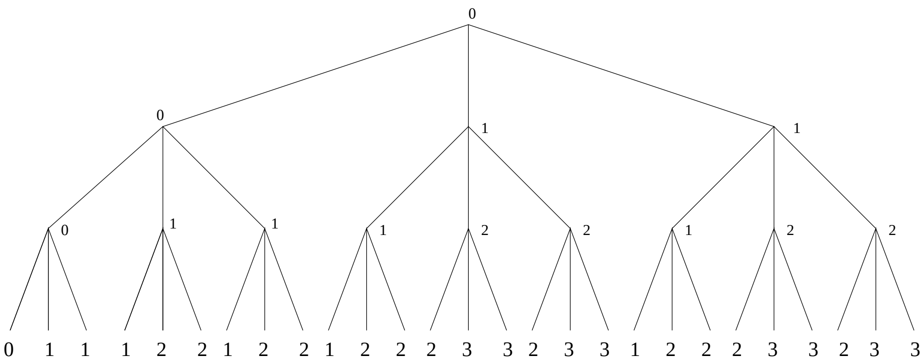

FIG. 1 – *Nombre de divergence accumulés en chaque nœud d'un arbre ternaire complet.*

L'algorithme complet qui lui correspond consiste à faire d'abord une recherche sans que plus de k choix de l'heuristique ne soient outrepassés, puis, si une solution n'a pas été trouvée, de recommencer la recherche en incrémentant la valeur de k (voir figure 2). Un inconvénient de LDS est la redondance en terme de génération de nœuds. Dans le cas où un arbre binaire de profondeur n ne contient pas de solution, si on considère qu'aucun élagage n'a lieu, alors LDS aura visité $\frac{n+2}{2}2^n$ feuilles au lieu de 2^n avec une DFS.

Il existe une autre façon de mettre en œuvre LDS: on peut aussi faire un algorithme incomplet en fixant une seule fois k, ce qui permet un développement partiel de l'arbre,

```
LDS-iteration(nœud, k):
    SI nœud est une solution ALORS retourne nœud
    s := Successeurs(nœud)
    SI s = ∅ ALORS retourne NIL
    SI k = 0 ALORS retourne LDS-iteration(fils-gauche(s),0)
    SINON resultat := LDS-iteration(fils-droit(s),k − 1)
                SI resultat ≠ NIL ALORS retourne resutat
                retourne LDS-iteration(fils-gauche(s),k)
```

```
LDS(nœud)
    POUR x de 0  la profondeur maximale
        resultat ; = LDS-iteration(nœud, x)
        SI resultat ≠ NIL ALORS retourne resultat
    retourne NIL
```

FIG. 2 – *LDS*

restreint aux zones que l'heuristique estime les plus prometteuses.

Dans leur article introduisant LDS [HG95], Harvey et Ginsberg font une comparaison théorique entre DFS et LDS sur un problème ayant beaucoup de solutions (une solution pour mille feuilles) et calculent que lorsque l'heuristique est très bonne (>95 % de bon choix) alors LDS a bien plus de chance que DFS de trouver une solution lors de l'itération qui accepte une seule divergence au maximum.

Par ailleurs, Korf a proposé une amélioration de LDS qu'il appelle *Improved LDS* (ILDS)[Kor96] qui permet à la recherche d'éviter de se diriger vers les feuilles de profondeur n dont la divergence est inférieure à celle de l'itération courante. Ainsi, chaque feuille n'est plus visitée qu'une seule fois (voir fig. 3 et 4), comme pendant une DFS. Par contre, la redondance subsiste pour les nœuds internes de l'arbre. Ainsi, un arbre binaire complètement développé par une DFS contient deux fois moins de nœuds que celui développé par une ILDS. Il faut en plus prendre en compte le fait qu'un échec peut survenir bien avant la profondeur maximale de l'arbre ce qui défavorise encore ILDS. Malgré tout, comme les feuilles sont examinées dans un ordre plus prometteur que celui d'une DFS, ILDS, encore plus que LDS, reste une bonne alternative à DFS pour les problèmes dont l'heuristique est suffisamment bonne. Comme l'amélioration de LDS que constitue ILDS est manifeste bien qu'elle ne modifie pas LDS de façon fondamentale, nous convenons, à partir de maintenant, de substituer le terme "LDS" à "ILDS". Sauf mention explicite l'invalidant, "LDS" désignera toujours une ILDS.

```
ILDS-iteration(nœud, k, profondeur):
    SI nœud est une solution ALORS retourne nœud
    s := Successeurs(nœud)
    SI s = ∅ ALORS retourne NIL
    SI profondeur > k ALORS retourne ILDS-iteration(fils-gauche(S,k,profondeur − 1)
    SI k > 0 ALORS retourne LDS-iteration(fils-droit(S),k − 1,profondeur − 1)
```

FIG. 3 – *La procédure d'itération de ILDS. La procédure appelant* ILDS-iteration *est la même que celle de LDS excepté qu'il faut spécifier la profondeur maximale de l'arbre lors de l'appel de* ILDS-iteration.

3 Justification théorique de LDS

LDS a déjà été justifié de façon théorique en utilisant le modèle abstrait de recherche heuristique de [HG95]. Dans ce modèle d'arbre binaire complet de profondeur n, un choix aléatoire de successeur a une probabilité constante m d'être une erreur (c'est-à-dire un choix de sous-arbre ne contenant pas de solution alors que l'autre sous-arbre

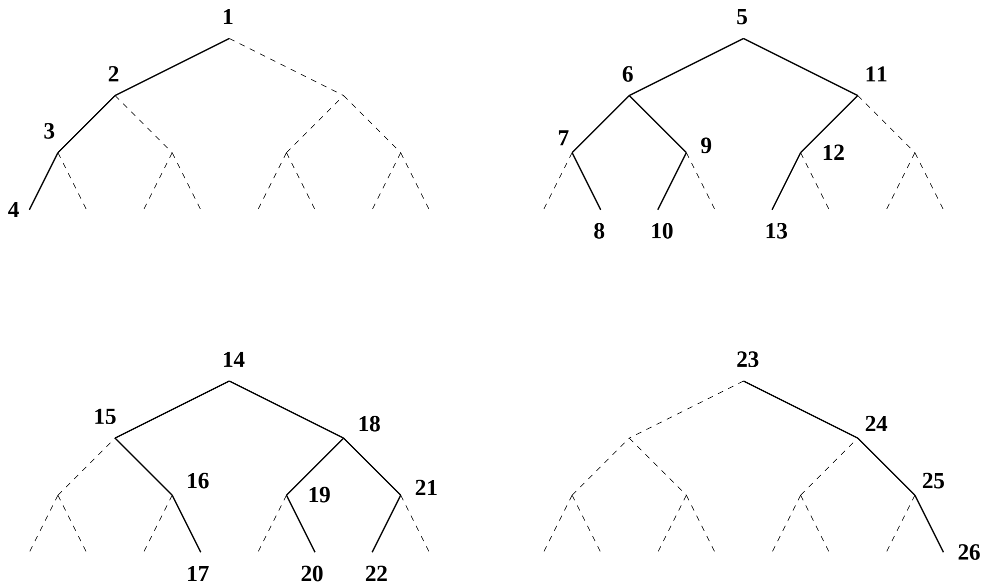

FIG. 4 – *Les différentes itérations de ILDS pour un arbre binaire complet de profondeur 3 et l'ordre dans lequel les nœuds sont générés.*

en a au moins une). Avec une heuristique, le premier successeur d'un noeud a la probabilité p d'avoir une solution dans son sous-arbre, avec $p > 1 - m$. Il a été vérifié expérimentalement que m pouvait être considéré comme constant [HG95]. Néanmoins, la probabilité p due à l'heuristique a été aussi supposée constante, ce qui est souvent faux en pratique et contredit l'hypothèse de h-progression - que nous définissons ci-dessous - sur laquelle s'appuient d'autres recherches arborescentes basées sur les divergences comme par exemple IDFS[Mes97] et DDS[Wal97]. Nous allons donc proposer un modèle de recherche heuristique qui prenne en compte la h-cohérence et la h-progression afin d'analyser la pertinence de l'ordre de visite des feuilles par LDS de façon plus fine. Ce modèle probabiliste et l'ordre partiel sur les feuilles qui en résulte a été introduit dans [PN99]. Nous allons en rappeler les points essentiels.

3.1 Le modèle de recherche heuristique

Afin de simplifier les explications sur le modèle, nous n'utiliserons que les arbres binaires complets de profondeur n. Cependant, le modèle est très facilement extensible aux arbres d'arité supérieure. Il suffirait de considérer que tous les choix heuristiques autres que le premier correspondent à une divergence d'importance égale.

Nous définissons α_i comme étant la probabilité que l'heuristique fasse le bon choix à la profondeur i, c'est-à-dire que la branche gauche de l'arbre mène à une solution, *sachant qu'au moins un des deux sous-arbres contient au moins une solution.* β_i est la probabilité de faire le bon choix quand on diverge du choix heuristique à la profondeur i, c'est-à-dire que la branche droite de l'arbre mène à une solution, *sachant qu'au moins un des deux sous-arbres contient au moins une solution.*

La probabilité P qu'une feuille L soit une solution est la probabilité que les n choix effectués pour constituer le chemin qui mène à cette feuille soient justes. Donc, nous avons :

P = Probabilite($choix_1$ est juste ET $choix_2$ est juste ... ET $choix_n$ est juste)
$P = \prod_{i=1}^{i=n}$ Probabilite($choix_i$ est juste $|\forall j < i, choix_j$ est juste)
$P = \prod_{i=1}^{i=n} p_i$, où $p_i = \alpha_i$ si le choix de l'heuristique a été respecté à la profondeur i et $p_i = \beta_i$ dans le cas contraire.

Nous faisons l'hypothèse que notre modèle de recherche heuristique est h-cohérent et h-progressif :

- h-cohérence. La probabilité que la branche gauche mène à une solution est toujours supérieure à celle de la branche droite : $\forall i \ \alpha_i \geq \beta_i$.
- h-progression. Il est toujours préférable d'avoir une divergence à faible profondeur plutôt qu'à une plus grande profondeur : $\forall i < j, \alpha_i \beta_j \leq \alpha_j \beta_i$.

3.2 Conditions pratiques de h-cohérence et h-progression

La condition de h-cohérence est facile à réaliser en pratique. Si une heuristique s'avère se tromper plus souvent qu'elle ne fait le bon choix à la profondeur i alors il suffit de prendre systématiquement l'opposé de ce choix à cette profondeur pour revenir à une h-cohérence.

Par contre, la h-progression ne va pas de soi. Elle dépend de l'accroissement de la qualité de l'heuristique avec la profondeur.

Nous estimons qu'il est courant que l'heuristique de choix de valeurs ait une qualité qui s'accroît avec la profondeur dès lors que cette heuristique est *dynamique*, c'est-à-dire qu'elle est calculée pendant la recherche arborescente au moment où il faut affecter une valeur à une variable sélectionnée.

Un exemple d'une telle heuristique est la version dynamique de l'heuristique des solutions potentielles (Max-promises value ordering heuristic [Gee92]) lorsqu'elle est utilisée avec une procédure de filtrage des domaines restant à instancier. L'heuristique des solutions potentielles ordonne les valeurs du domaine de la variable courante à instancier de telle façon que les valeurs disposant d'un espace de recherche plus grand soient choisies en premier. Le nombre de solutions potentielles pour une valeur v est le produit du cardinal des domaines des variables restant à instancier desquels on a retirées les valeurs incompatibles avec v.

Dans le cas d'une utilisation statique de l'heuristique des solutions potentielles, les domaines sont ordonnés avant la recherche arborescente. Comme chaque domaine dispose d'informations équivalentes, la qualité de l'ordre des valeurs est identique pour chaque variable. Par contre, l'utilisation dynamique de cette heuristique a un effet d'amélioration de la qualité de l'ordre des valeurs. Comme une procédure de filtrage réduit graduellement les domaines des variables restant à instancier au fur et à mesure que des variables sont instanciées, l'heuristique devient de plus en plus sure quand la profondeur croît parce qu'elle évite de prendre en compte des valeurs qui ont été éliminées. Ainsi, le nombre de solutions potentielles calculé devient de plus en plus proche de la vraie valeur.

3.3 Un ordre partiel sur les feuilles

Si nous connaissions toutes les valeurs α_i et β_i, nous pourrions définir un algorithme qui visite les feuilles dans un ordre croissant de probabilité. Ce n'est pas le cas en général mais il est malgré tout possible de comparer certaines probabilités qu'ont des feuilles d'être solution dès lors qu'on considère un modèle probabiliste h-cohérent et h-progressif. Par exemple, si la feuille L1 a la probabilité $P_1 = \alpha_1\beta_2\alpha_3$ d'être une solution et la feuille L2 a la probabilité $P_2 = \alpha_1\alpha_2\beta_3$ d'être une solution alors nous savons que $P_1 \geq P_2$ car $\beta_2\alpha_3 \geq \alpha_2\beta_3$. Cependant, si $P_1 = \alpha_1\alpha_2\beta_3$ et $P_2 = \beta_1\beta_2\alpha_3$ alors nous ne pouvons établir ni que $P_1 \leq P_2$ ni que $P_1 \geq P_2$.

Définition 3.1 *P-consistance*
Un algorithme de recherche A est P-consistant *si et seulement si pour toutes feuilles L_k et L_{k+1} générées consécutivement par A, il existe un ensemble de valeurs α_i et β_i h-cohérent et h-progressif tel que $P_k \geq P_{k+1}$, P_k et P_{k+1} étant les probabilités de L_k et L_{k+1} d'être une solution.*

S'assurer qu'un algorithme de recherche est P-consistant est une manière de garantir un ordre de visite des feuilles compatible avec les hypothèses de h-cohérence et de h-progression de l'heuristique de choix de valeur lorsque les valeurs des α_i et des β_i sont inconnues. Néanmoins, cette définition de la P-consistance rend peu pratique la preuve de P-consistance d'un algorithme. C'est pourquoi nous avons défini un ordre partiel sur les feuilles qui va nous permettre de les comparer de façon plus opérationnelle.

Définition 3.2 *Traîne d'une branche*
La traîne de longueur l d'une branche est la séquence des l derniers choix de valeurs effectués avant d'atteindre sa feuille (de profondeur n).

Définition 3.3 *L'ordre partiel $\prec$*
Soit C et C' deux branches allant de la racine d'un arbre de recherche à des feuilles L et L' de profondeur n. $L \prec L'$ si et seulement si, quelle que soit l, la traîne de C de longueur l contient un nombre de divergences inférieur ou égal à la traîne de C' de même longueur.

La relation $\prec$ est un ordre partiel : elle est trivialement réflexive et transitive. Elle est aussi antisymétrique : si $L \prec L'$ et $L' \prec L$ alors on démontre facilement par récurrence, en examinant les traînes des branches menant à L et L' dans l'ordre croissant de leur longueur, qu'elles sont constituées des mêmes choix de valeur, ce dont on conclut que $L = L'$.

L'ordre partiel $\prec$ est défini sur l'ensemble des feuilles d'un arbre binaire de recherche complet. Le théorème suivant lie cet ordre partiel avec notre modèle probabiliste.

Proposition 3.1 *Soit deux branches partant de la racine et menant respectivement aux feuilles L et L' de profondeur n avec les probabilités P et P' d'être solution dans un modèle probabiliste h-cohérent et h-progressif.*
$\forall\{(\alpha_i,\beta_i), i = 1..n\}$ *h-cohérent et h-progressif,* $P \geq P' \Leftrightarrow L \prec L'$.

Preuve. Voir [PN99].

Maintenant, nous pouvons donner une manière opérationnelle de controler la P-consistance d'un algorithme.

Définition 3.4 *Respect de l'ordre partiel* $\prec$
Un algorithme de recherche arborescente respecte *l'ordre partiel* $\prec$ *si et seulement si toutes branches générées consécutivement* C_k *et* C_{k+1} *menant aux feuilles* L_k *et* L_{k+1} *sont telles que* $L_{k+1} \prec L_k$ *est faux, c'est-à-dire : au moins une traîne de* C_{k+1} *a plus de divergences qu'une traîne de* C_k *de même longueur. On dira que l'algorithme* viole *l'ordre partiel* $\prec$ *s'il ne le respecte pas.*

Proposition 3.2 *Un algorithme de recherche arborescente est P-consistant si et seulement si il respecte l'ordre partiel* $\prec$*.*

Preuve. Voir [PN99].

LDS ne respecte pas l'ordre partiel $\prec$. Il suffit de comparer les deux premières feuilles L et L' atteignables lors de l'itération 1 (celle qui n'autorise qu'une divergence) : la divergence de la première feuille a lieu en profondeur n tandis que celle de la deuxième a lieu en profondeur $n-1$. Leur probabilité d'être solution sont $P = \prod_{i=1}^{i=n-2} \alpha_i.\alpha_{n-1}.\beta_n$ et $P' = \prod_{i=1}^{i=n-2} \alpha_i.\beta_{n-1}.\alpha_n$ et comme $\alpha_{n-1}.\beta_n \leq \beta_{n-1}.\alpha_n$ sous l'hypothèse de la h-progression alors $P \leq P'$.

Nous allons voir qu'il suffit de modifier très légérement LDS pour en faire un algorithme P-consistant.

4 Variantes P-consistantes de LDS

Dans un premier temps, nous allons présenter RLDS, la recherche à divergence limitée inversée, qui est P-consistante pour les arbres d'arité 2. Puis, nous verrons que son adaptation pour des arbres d'arité supérieure à 2 pose des problèmes. Nous définissons alors DWDS, une variante simple de LDS qui est P-consistante et qui s'adapte facilement aux arbres d'arité supérieure à 2.

4.1 La recherche à divergence limitée inversée

La recherche à divergence limitée inversée (Reverse LDS, RLDS) ne diffère de LDS que par le choix de valeur à examiner en premier lors d'une itération de LDS : le fils droit est choisi avant le fils gauche (voir fig. 5). Ceci permet de générer les branches contenant le même nombre de divergences dans un ordre où celles qui ont des divergences le plus en haut de l'arbre sont générées les premières.

Proposition 4.1 *RLDS est P-consistante.*

Preuve.
Soit C_i et C_{i+1} deux branches consécutivement explorées par RLDS. Il y a deux possibilités :

- C_{i+1} a une divergence de plus que C_i.

```
RLDS-iteration(node, k, depth):
1    IF node is a solution THEN return node
2    s := Successors(node)
3    IF s = ∅ THEN return NIL
4    result := NIL
5    IF k > 0 THEN result := RLDS-iteration(right-son(S),k − 1,depth − 1)
6    IF result = NIL AND depth > k
7        THEN result := RLDS-iteration(left-son(S),k,depth − 1)
8    return result
```

```
RLDS(node, max-depth)
1    FOR x from 0 to max-depth
2        result := RLDS-iteration(node, x, max-depth)
3        IF result ≠ NIL THEN return result
4    return NIL
```

FIG. 5 – *RLDS*

- C_i et C_{i+1} ont le même nombre de divergences. C_i et C_{i+1} partent de la racine et font les mêmes choix de valeurs jusqu'à la profondeur d où leur choix diffère : C_{i+1} respecte le choix de l'heuristique tandis que C_i ne le fait pas. Ainsi, la traîne de C_{i+1} débutant en profondeur $d+1$ a une divergence de plus que celle de C_i débutant en profondeur $d+1$.

Dans les deux cas, il existe une traîne de C_{i+1} contenant une divergence de plus qu'une traîne de C_i de même longueur donc la proposition 3.2 nous permet de conclure que RLDS est P-consistante.□

4.2 Généralisation de RLDS aux arbres d'arité quelconque

S'il existe des nœuds d'arité supérieure à 2 alors il faut adapter l'algorithme de RLDS. Nous nous placerons dans le cas le plus simple ou on considère que toute alternative au premier choix a l'égale probabilité β_i de mener à une solution. Or, même dans ce cas simple la modification de l'algorithme n'est pas triviale. Il ne suffit pas de remplacer la ligne 5 de l'algorithme par une boucle d'appel de RLDS_iteration() sur tous les fils divergents. En effet, les sous-arbres issus des fils divergents d'un même nœud ont la même distribution de probabilité que leurs feuilles soient solutions. Il existe donc, par exemple, des branches d'un sous-arbre issu du deuxième fils divergent qui doivent être explorées avant celles du sous-arbre du premier fils divergent. Il en résulte que générer toutes les branches de même divergence à l'intérieur d'un sous-arbre avant de passer au suivant ne maintient pas la P-consistance. Par exemple, considérons un arbre de profondeur 3 et numérotons les choix de valeurs à partir de 0. La branche (0,2,1) correspond au choix de la première valeur à la profondeur 1, de la troisième valeur à la profondeur 2 et de la deuxième valeur à la profondeur 3. Pour que RLDS soit P-consistante, il faudrait que, lors de l'itération 2, la branche (2,2,0) soit examinée avant la branche (1,0,2). Or, en l'état, RLDS examine d'abord toutes les branches de type (1,x,y) (telles que soit $x = 0$, soit $y = 0$) avant d'examiner les branches de type (2,x,y). Une généralisation de RLDS aux arbres d'arités quelconque consisterait à générer successivement toutes les branches qui ont les mêmes divergences aux mêmes profondeurs, ce qui pose un problème majeur de redondance dans la génération des nœuds internes de l'arbre. Il faudrait faire un algorithme qui effectue $2^n + 1$ itérations[1] (au lieu des $n + 1$ de LDS) pour être sûr d'examiner toutes les branches qui ont les mêmes divergences aux mêmes profondeurs avant de passer aux autres.

1. C'est le nombre de possibilités d'avoir des divergences sur n profondeurs

Le passage à des arbres d'arité quelconque nécessite une adaptation de la définition de l'ordre partiel $\prec$ à ce nouveau contexte. En effet, lorsque deux feuilles distinctes L et L' ont les mêmes divergences aux mêmes profondeurs, on a $L \prec L'$ et $L' \prec L$, ce qui empêche la relation $\prec$ d'être anti-symétrique et donc d'être un ordre partiel. C'est pourquoi nous élargissons maintenant la définition de l'ordre partiel $\prec$.

Définition 4.1 *L'ordre partiel $\prec$ (élargi aux arbres d'arité quelconque)*
Soit E et E' deux ensembles regroupant toutes les feuilles dont chacune possède les mêmes divergences aux mêmes profondeurs que les autres feuilles de leur ensemble. $E \prec E'$ si et seulement si, quel que soit l, la traîne des feuilles de E de longueur l contient un nombre de divergences inférieur ou égal à la traîne des feuilles de E' de même longueur.

Remarquons que cette nouvelle définition reste utilisable pour les arbres binaires où les ensembles de feuilles sont des singletons. Néanmoins, il nous faut redéfinir ce que signifie maintenant de respecter l'ordre partiel $\prec$.

Définition 4.2 *Respect de l'ordre partiel $\prec$*
Un algorithme de recherche arborescente respecte *l'ordre partiel $\prec$ si et seulement si toutes branches générées consécutivement C_k et C_{k+1} menant aux feuilles L_k et L_{k+1} sont telles que :*

- *soit L_k et L_{k+1} ont les mêmes divergences aux mêmes profondeurs.*
- *soit L_k et L_{k+1} n'ont pas toutes leurs divergences aux mêmes profondeurs et les ensembles disjoints E et E' de feuilles ayant les mêmes divergences aux mêmes profondeurs auxquelles appartiennent respectivement L_k et L_{k+1} sont tels que $E' \prec E$ est faux.*

La proposition 3.2 liant la P-consistance avec le respect de l'ordre partiel subsiste. En effet, dans le cas nouveau où deux feuilles L_k et L_{k+1} ont les mêmes divergences aux mêmes profondeurs, l'ordre partiel est respecté par définition et les probabilités P_k et P_{k+1} de ces feuilles d'être solution sont égales.

4.3 La recherche à divergence pondérée par sa profondeur

Comme la transposition de RLDS aux arbres d'arité quelconque semble inefficace à cause d'une forte redondance dans la génération des nœuds internes, nous proposons un autre algorithme : la recherche à divergence pondérée par sa profondeur (Depth-Weighted Discrepancy Search, DWDS). Plutôt que de limiter le nombre de divergences, elle limite la somme des profondeurs des divergences. Ainsi, à une somme de profondeur de divergences fixée correspondent des combinaisons de divergences telles que plus les divergences sont profondes, moins elles sont nombreuses. Nous allons voir que DWDS est P-consistante.

Pour que DWDS soit efficace, il faut que l'algorithme ne génère que des nœuds susceptibles a priori de mener à des feuilles correspondant à la somme des profondeurs de divergence qui a été fixée. Or, déterminer si un nœud de profondeur $k-1$ mène potentiellement à une feuille dont la somme des divergences est S revient à déterminer s'il existe un entier e tel que S est la somme de e entiers distincts de l'intervalle $[k;n]$. Or, comme l'ensemble des sommes possibles de e entiers disctincts de l'intervalle $[k;n]$ regroupe toutes les valeurs comprises entre la somme des e premiers entiers et la somme des e derniers entiers de l'intervalle $[k;n]$, cette condition est équivalente à :
$\exists e \in \mathbb{N}, \sum_{i=k}^{i=k+e-1} i \leq S \leq \sum_{i=n-e+1}^{i=n} i$.

En résolvant l'inéquation on obtient :
$\exists e \in \mathbb{N},(2n+1)^2 \geq 8S$ et $\frac{2n+1-\sqrt{(2n+1)^2-8S}}{2} \leq e \leq \frac{-2k+1+\sqrt{(2k-1)^2+8S}}{2}$

En appelant E$^-$ (resp. E$^+$) la fonction d'arrondi à l'entier inférieur (resp. supérieur), on obtient finalement que S est décomposable en entiers distincts de l'intervalle $[k;n]$ si et seulement si
$(2n+1)^2 \geq 8S$ et $E^+(\frac{2n+1-\sqrt{(2n+1)^2-8S}}{2}) \leq E^-(\frac{-2k+1+\sqrt{(2k-1)^2+8S}}{2})$.

Nous avons donc une relation évaluable en temps constant qui nous permet de rejeter tout nœud de l'arbre de recherche qui ne peut pas mener à une solution dont la somme des profondeurs de divergences égale une constante S.

```
DWDS-iteration(nœud, S, prof):
    SI nœud est une solution ALORS retourne nœud
    F := Les_fils(nœud)
    SI F = ∅ ALORS retourne NIL
    SI decomposable(S, prof + 1, n) ALORS retourne DWDS-iteration(premier(F),S,profondeur + 1)
    TANTQUE F ≠ ∅
       SI decomposable(S − prof, prof + 1, n) ALORS retourne DWDS-iteration(premier(F),S − prof,prof + 1)
       F := F\ premier(F)
    retourne NIL
```

```
DWDS(nœud)
    POUR x de 0  n.(n + 1)/2
         resultat := DWDS-iteration(nœud, x, 0)
         SI resultat ≠ NIL ALORS retourne resultat
    retourne NIL
```

FIG. 6 – *DWDS*

Proposition 4.2 *DWDS est P-consistante.*

Preuve.
Soient L et L' deux feuilles distinctes générées consécutivement pendant une DWDS. Soient E et E' les ensembles de feuilles ayant les mêmes divergences aux mêmes profondeurs auxquelles appartiennent respectivement L et L'. On note d_i pour signifier une divergence ($d_i = 1$) ou son absence ($d_i = 0$) à la profondeur i. Si $\forall i, d_i = d'_i$ alors L et L' appartiennent au même ensemble ($E = E'$) et l'ordre partiel est respecté par définition. Examinons maintenant le cas où L et L' sont telles que $\exists i, d_i \neq d'_i$. DWDS est telle que quelles que soient les feuilles consécutives L et L' qu'elle génère, on a : $\sum_{i=1}^{i=n} id_i \leq \sum_{i=1}^{i=n} id'_i$ (1). Raisonnons par l'absurde et posons que $E' \prec E$. $E' \prec E \Rightarrow \forall l \leq n, \sum_{i=l}^{i=n} d'_i \leq \sum_{i=l}^{i=n} d_i$ (2)
Comme $\exists i, d_i \neq d'_i$, on sait qu'au moins une traîne de L' a strictement moins de divergences que celle de L de même longueur, c'est-à-dire : $\exists l \leq n, \sum_{i=l}^{i=n} d'_i < \sum_{i=l}^{i=n} d_i$ (3).
En faisant la somme des inégalités (larges) de (2) pour l variant de 1 à n mais en tenant compte de l'inégalité stricte de (3) on obtient l'inégalité stricte :
$\sum_{l=1}^{l=n} \sum_{i=l}^{i=n} d'_i < \sum_{l=1}^{l=n} \sum_{i=l}^{i=n} d_i$.
En utilisant la formule $\sum_{l=1}^{l=n} \sum_{i=l}^{i=n} x_i = \sum_{i=1}^{i=n} ix_i$, on simplifie la relation ainsi : $\sum_{i=1}^{i=n} id'_i < \sum_{i=1}^{i=n} id_i$. Or, cette relation contredit la relation (1) donc l'hypothèse $E' \prec E$ est fausse. Que $E' \prec E$ soit faux implique que DWDS respecte l'ordre partiel $\prec$.
Comme on a montré que dans tous les cas DWDS respectait l'ordre partiel, la proposition 3.2 nous permet de conclure que DWDS est P-consistante. □

DWDS et RLDS n-aire sont donc P-consistante mais, alors que RLDS semble nécessiter au pire 2^n itérations, le nombre d'itérations de DWDS dans le pire des cas est $\frac{n.(n+1)}{2} + 1$. Cependant, la redondance dans la génération des nœuds internes de DWDS est supérieure à celle de LDS.

5 Variantes avec intervalle de divergences

Le critère de la P-consistance n'est pas suffisant pour déterminer les efficacités respectives des algorithmes de recherche arborescente. Le problème principal des algorithmes basés sur les divergences, qui peut les empêcher d'être meilleurs que DFS, est la redondance dans la visite des nœuds internes de l'arbre de recherche. Afin de pouvoir concurrencer DFS, ils peuvent nécessiter un aménagement qui permet de choisir un compromis entre l'altération partielle de la P-consistance et la réduction de la redondance. Une solution consiste à prendre un intervalle de divergences plutôt qu'une seule divergence à chaque itération de l'algorithme.

5.1 La recherche à divergence bornée

La recherche à divergence bornée (DBDFS)[BP00] est une variante très simple de LDS. Au lieu de se restreindre à la visite de branches ayant exactement le même nombre de divergences lors de chaque itération de LDS, on examinera les branches ayant leur nombre de divergences appartenant à un intervalle $[(i-1).w,i.w-1]$, où i est le numéro de l'itération et w la longueur de l'intervalle. Encore une fois, le but est de réduire la redondance au détriment de la qualité de l'ordre de visite des branches. Afin de simplifier les notations, nous appellerons LDS(w), une DBDFS dont la taille de l'intervalle de divergences permises est w.

Nous allons calculer $N(n,w)$, le nombre maximum de noeuds qui seraient générés par LDS(w) pour un arbre binaire complet de profondeur n. Un noeud situé à la profondeur i et ayant accumulé k divergences sera visité pour les divergences allant de k à $n-i+k$, soit $E(\frac{n-i}{w})+1$ fois, $E(x)$ étant la partie entière de x. Comme à cette profondeur, il y aura au maximum 2^i noeuds, on a :

$$\begin{aligned}
N(n,w) &= \textstyle\sum_{i=0}^{i=n} 2^i(E(\frac{n-i}{w})+1)\\
&= 2^{n+1}-1+\textstyle\sum_{i=0}^{i=n} 2^i E(\frac{n-i}{w})\\
&= 2^{n+1}-1+2^n\textstyle\sum_{i=0}^{i=n} 2^{-i} E(\frac{i}{w})\\
&= 2^{n+1}-1+2^n\textstyle\sum_{i=0}^{i=\frac{n+1}{w}-1}\sum_{j=0}^{i=w-1} 2^{-iw-j} E(\frac{iw+j}{w})\\
&= 2^{n+1}-1+2^n\textstyle\sum_{i=0}^{i=\frac{n+1}{w}-1}\sum_{j=0}^{i=w-1} 2^{-iw-j}.i\\
&= 2^{n+1}-1+2^n\textstyle\sum_{i=0}^{i=\frac{n+1}{w}-1} i.2^{-iw}\sum_{j=0}^{i=w-1} 2^{-j}\\
&= 2^{n+1}-1+2^n(2-\frac{1}{2^{w-1}})\textstyle\sum_{i=0}^{i=\frac{n+1}{w}-1} i(2^{-w})^i
\end{aligned}$$

Et en appliquant la formule $\sum_{i=0}^{i=k} ik^i = \frac{(kx-k-1)x^{k+1}+x}{(x-1)^2}$, on obtient :

$$\begin{aligned}
N(n,w) &= 2^{n+1}-1+2^n(2-\frac{1}{2^{w-1}})(\frac{((\frac{n+1}{w}-1)2^{-w}-(\frac{n+1}{w}-1)-1)(2^{-w})^{\frac{n+1}{w}}+2^{-w}}{(2^{-w}-1)^2})\\
&= 2^{n+1}-1+\frac{2^{n+1}-1}{2^w-1}-\frac{n+1}{w}
\end{aligned}$$

Alors que par rapport à DFS, LDS a le double de noeuds ($N(n,1) \simeq 2N(n,n+1)$), LDS(2) n'en a qu'un tiers de plus et LDS(3) n'en a qu'un septième de plus.

On sait que LDS ne surpasse DFS que lorsque l'heuristique est très bonne et qu'on effectue ainsi très peu d'itérations. Par contre, si on n'utilise pas d'heuristique, LDS(w) n'apporte rien en terme d'ordre de visite des branches donc DFS est la méthode la plus efficace car elle n'est pas redondante. Comme LDS(w) est une généralisation de DFS et

	1020	1030	1040	1050	1060
MAC	**8.35**	**11.1**	**23.7**	**35.4**	**36.2**
LDS(1)	1.68	**3.64**	**5.05**	**7.53**	**8.66**
LDS(2)	**1.53**	4.15	5.45	8.55	14.9
LDS(3)	3.81	4.70	6.89	9.91	9.80
LDS(4)	4.67	7.32	10.0	15.0	18.6
LDS(5)	6.41	9.32	16.1	23.0	28.8
DWDS(1)	1.29	1.87	2.66	3.20	7.60
DWDS(2)	1.27	1.90	2.74	3.21	6.95
DWDS(3)	**1.14**	1.88	**2.44**	3.20	**6.77**
DWDS(4)	1.31	1.99	2.88	3.33	6.94
DWDS(5)	1.16	**1.78**	2.82	**3.18**	7.33
DWDS(10)	1.46	1.79	2.68	3.91	6.23
DWDS(40)	4.19	7.71	9.03	15.9	19.4

TAB. 1 – *Nombre de branches générées (en milliers) pour 100 instances de problèmes aléatoires (*$n = 50$*,* $m = 8$*,* $p1 = k/1225$*,* $p2 = 5/64$*), k variant de 1020 à 1060.*

de LDS, quelle que soit la qualité de l'heuristique dont on dispose, on sait qu'il existe un w tel que LDS(w) sera autant ou plus efficace que DFS. En pratique, on fixera w à une valeur d'autant plus petite que la qualité de l'heuristique sera bonne.

La version de DWDS avec intervalles de somme de profondeurs de divergence est aussi facile à réaliser que celle de LDS et ne pose pas de problèmes particuliers. Notons simplement que DWDS(w) est une généralisation de DFS et de DWDS car DWDS($\frac{n.(n+1)}{2}$) = DFS et DWDS(1) = DWDS.

6 Expérimentations

Les expérimentations ont été effectuées avec des CSP aléatoires de la bibliothèque C++ de Hulubei[Hul99]. Les quatre paramètres du CSP aléatoire (n, le nombre de variables, m, la taille de chaque domaine, $p1$, la densité du graphe de contraintes et $p2$, la difficulté des contraintes) ont été choisis de façon à rester à gauche du pic de complexité. Pour chaque paramétrage, nous avons exécuté différents algorithmes sur les mêmes 100 instances de problèmes dont chacun avait au moins une solution. Tous les algorithmes sont basés sur MAC (ils maintiennent la cohérence d'arc [SF94]), utilisent l'heuristique *min-domain/deg* [BR96] et l'heuristique de choix de valeur dynamique *Max-Promises* [Gee92].

Sur ces tests préliminaires, on remarque que le nombre de branches générées par DWDS est plus petit que pour LDS (voir tab. 1). Ceci confirme que DWDS sélectionne mieux que LDS les branches les plus susceptibles d'être des solutions. Par contre, les temps de résolution de DWDS ne sont pas bons (voir tab. 2). Ils peuvent être meilleur que MAC si l'intervalle des sommes de profondeurs de divergences est conséquent mais, pour l'instant, nous n'avons pas trouvé de cas où DWDS était plus rapide que LDS avec un intervalle optimal. Il est possible que les conditions particulières de nos tests (CSP aléatoires, maintien de la cohérence d'arc) font que l'heuristique de choix de valeur dynamique choisie n'est pas assez bonne pour réduire suffisamment le nombre de branches

	1020	1030	1040	1050	1060
MAC	**79.1**	**102**	**211**	**330**	**303**
LDS(1)	92.6	155	229	312	351
LDS(2)	79.9	138	189	259	378
LDS(3)	76.9	**99.2**	155	243	**220**
LDS(4)	**63.1**	101	**136**	**230**	258
LDS(5)	71.4	101	177	277	310
DWDS(1)	188	259	360	420	891
DWDS(2)	177	248	352	400	787
DWDS(3)	153	228	294	374	714
DWDS(4)	166	235	334	378	727
DWDS(5)	140	204	306	351	725
DWDS(10)	116	139	202	291	468
DWDS(40)	**78.2**	**138**	**156**	**276**	**329**

TAB. 2 – *Temps total (en secondes) pour 100 instances de problèmes aléatoires (*$n = 50$*,* $m = 8$*,* $p1 = k/1225$*,* $p2 = 5/64$*),* k *variant de 1020 à 1060.*

de telle sorte que les temps de résolution de DWDS deviennent inférieurs à ceux de LDS.

Ces résultats confirment que le seul critère de P-consistance ne suffit pas pour obtenir des algorithmes rapides. Nous savions que la redondance dans la génération des nœuds freinait l'efficacité des algorithmes basés sur les divergences. La nécéssité de faire fonctionner ces algorithmes avec des fenêtres de divergences s'est vue confirmée. Par ailleurs, en examinant le nombre de nœuds générés et le nombre de contraintes testées sur nos expérimentations, nous avons remarqué qu'à nombre de nœuds générés égal, LDS faisait plus de tests de contraintes que MAC et de même que DWDS faisait plus de tests de contraintes que LDS. Ceci s'explique ainsi : ce sont les nœuds situés le plus près de la racine qui sont le plus souvent regénérés. Or, c'est sur ceux-ci que sont effectués le plus de tests de contraintes, surtout si l'intensité du filtrage (ici, la cohérence d'arc) est élevée, et donc les plus coûteux en temps. Il en découle une idée simple d'amélioration de l'efficacité de LDS et DWDS : mémoriser les nœuds générés jusqu'à une certaine profondeur fixe afin de ne pas avoir à les regénérer lors d'une itération ultérieure. De cette façon, une partie non négligeable du nombre de test de contraintes redondants pourrait être évitée.

7 Conclusion

Nous avons présenté plusieurs variantes de LDS : RLDS et DWDS, qui modifient l'ordre de visite des nœuds de façon à ce que la recherche arborescente soit P-consistante, c'est-à-dire conforme aux propriétés standard de h-cohérence et de h-progression liées à l'utilisation d'une heuristique de choix de valeurs dynamique. Sachant que le respect du critère de P-consistance entraîne une génération redondante des nœuds internes de l'arbre, nous avons présenté les variantes de LDS et de DWDS qui permettent une réduction de cette redondance contre une certaine altèration dans l'ordre de visite des nœuds.

Nos premières expérimentations montrent que si DWDS(w) génère bien moins de branches que LDS(w) comme c'était prévu, cela ne suffit pas à rendre DWDS(w) plus rapide. Des expérimentations plus larges et plus poussées seront nécessaires pour vérifier si cette tendance se confirme généralement ou pas. Diminuer la redondance des tests de contraintes en mémorisant les nœuds peu profonds de l'arbre est une piste d'amélioration de ces algorithmes.

Références

[BP00] J.C. Beck and L. Perron. Discrepancy-Bounded Depth-First Search. In *Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CP-AI-OR'00)*, 2000.

[BR96] C. Bessiere and J. C. Regin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proc. of CP'96*, 1996.

[Gee92] P. A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the* 10^{th} *European Conference on Artificial Intelligence*, 1992.

[HG95] W. D. Harvey and M. L. Ginsberg. Limited Discrepancy Search. In *Proceedings of* 14^{th} *International Joint Conference on Artificial Intelligence*, pages 607–613, 1995.

[Hul99] T. Hulubei. *The CSP Library.* University of New Hampshire, 1999. http://www.cs.unh.edu/ tudor/csp/.

[Kor96] R. Korf. Improved limited discrepancy search. In *Proceedings of the* 13^{th} *AAAI Conference*, 1996.

[Mes97] P. Meseguer. Interleaved Depth-First Search. In *Proceedings of the* 15^{th} *International Joint Conference on Artificial Intelligence*, pages 1382–1387, 1997.

[PN99] N. Prcovic and B. Neveu. Ensuring a relevant visiting order of the leaf nodes during a tree search. In *Proceedings of the CP'99 Conference*, pages 361–374, 1999.

[SF94] D. Sabin and E. C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of ECAI'94*, pages 125–129, 1994.

[Wal97] T. Walsh. Depth-bounded discrepancy search. In *Proceedings of the* 15^{th} *International Joint Conference on Artificial Intelligence*, 1997.