Distribution en escalier pour le développement en parallèle de l'arbre de recherche des CSP

Nicolas Prcovic

Rapport de Recherche CERMICS 97.93

Février 97

Distribution en escalier pour le développement en parallèle de l'arbre de recherche des CSP

Nicolas Prcovic

Groupe Contraintes du CERMICS-INRIA Sophia Antipolis
2004, Route des Lucioles, B.P. 93
06902 Sophia Antipolis Cedex

Nicolas.Prcovic@sophia.inria.fr

Résumé

Nous présentons une nouvelle méthode de parallélisation de la recherche arborescente d'une solution d'un problème de satisfaction de contraintes, guidée par la volonté de garantir une accélération supérieure à une borne fixée. Elle consiste à choisir dynamiquement la profondeur à partir de laquelle les sous-arbres seront distribués grâce à un critère qui prend en compte une évaluation heuristique du nombre de noeuds du sous-arbre à paralléliser et qui estime ainsi la taille maximum de l'espace de recherche exploré. Nous montrons que cette technique est susceptible d'améliorer l'accélération.

Abstract

We present a new parallel tree search method for finding one solution to a constraint satisfaction problem that aims at guaranteeing a speedup greater than a fixed bound. It consists in choosing dynamically the depth of the subtrees to distribute. This choice is based on a criterion which takes into account a heuristic valuation of the number of nodes of the subtree to parallelize in order to estimate the maximum size of the search space. We show this method is likely to increase the speedup.

1 Introduction

Les problèmes de satisfaction de contraintes (CSP) étant NP-complets, ils deviennent souvent rapidement intraitables lorsque leur taille grandit. La parallélisation de la recherche d'une solution est un moyen de diviser le temps de résolution par un facteur important et ainsi de permettre l'obtention d'une solution dans un délai beaucoup plus raisonnable qu'auparavant.

Nous nous intéressons ici à la parallélisation de la recherche en profondeur d'abord (en anglais : Depth-First Search ou DFS) effectuée par l'algorithme Backtrack lorsqu'on désire obtenir une seule solution. Lorsqu'il s'agit de trouver toutes les solutions d'un problème, ce qui implique de développer l'arbre de recherche en entier, on arrive à obtenir des performances tout à fait satisfaisantes : l'accélération est proche de la linéarité, c'est-à-dire que le temps de résolution est pratiquement divisé par le nombre de processus engagés, tant en théorie [KGR94, Prc96] qu'en pratique [FM87, FK88, RKR87]. Par contre, quand on ne souhaite trouver qu'une seule solution ou bien la meilleure des solutions, on se trouve confronté à une difficulté supplémentaire : certains processus tournant en parallèle risquent d'explorer une partie de l'espace de recherche qui n'aurait pas été examinée lors d'une recherche séquentielle, ce qui peut induire un surcroît de travail inutile et donc une perte d'efficacité parfois majeure. Nous allons présenter dans cet article une méthode de parallélisation qui cherche à minimiser autant que possible ces risques tout en gardant peu élevé le surcoût dû aux communications interprocessus.

La section 2 donne quelques définitions sur les CSP et le parallélisme, la section suivante rappelle le fonctionnement du backtrack parallèle et la raison pour laquelle on peut obtenir une accélération quasi-linéaire lorsqu'on recherche toutes les solutions. En section 4, nous indiquons les méthodes généralement utilisées pour la recherche d'une seule solution et nous expliquons la nôtre. Puis, nous présentons les tests que nous avons effectués sur une série de problèmes aléatoires. Finalement, nous concluons en indiquant quelles perspectives concernant les problèmes d'optimisation peuvent découler de nos travaux.

2 Définitions et notations

2.1 Définition d'un CSP

Un CSP est défini par un triplet (X, D, C) avec : $X = \{x_1, ..., x_n\}$ un ensemble de n variables, $D = \{D_1, ..., D_n\}$ où D_i est le domaine fini dans lequel x_i peut prendre ses valeurs et C l'ensemble des contraintes sur les variables, qui définit la compatibilité des affectations de variables entre elles. Nous définissons d comme étant le cardinal du plus grand domaine.

Une solution est une affectation de toutes les variables avec une valeur de leur domaine qui ne viole aucune contrainte. L'algorithme Backtrack permet une recherche systématique de toutes les solutions en générant en profondeur d'abord un arbre de recherche de profondeur maximale égale au nombre de variables.

2.2 La résolution distribuée sur plusieurs machines

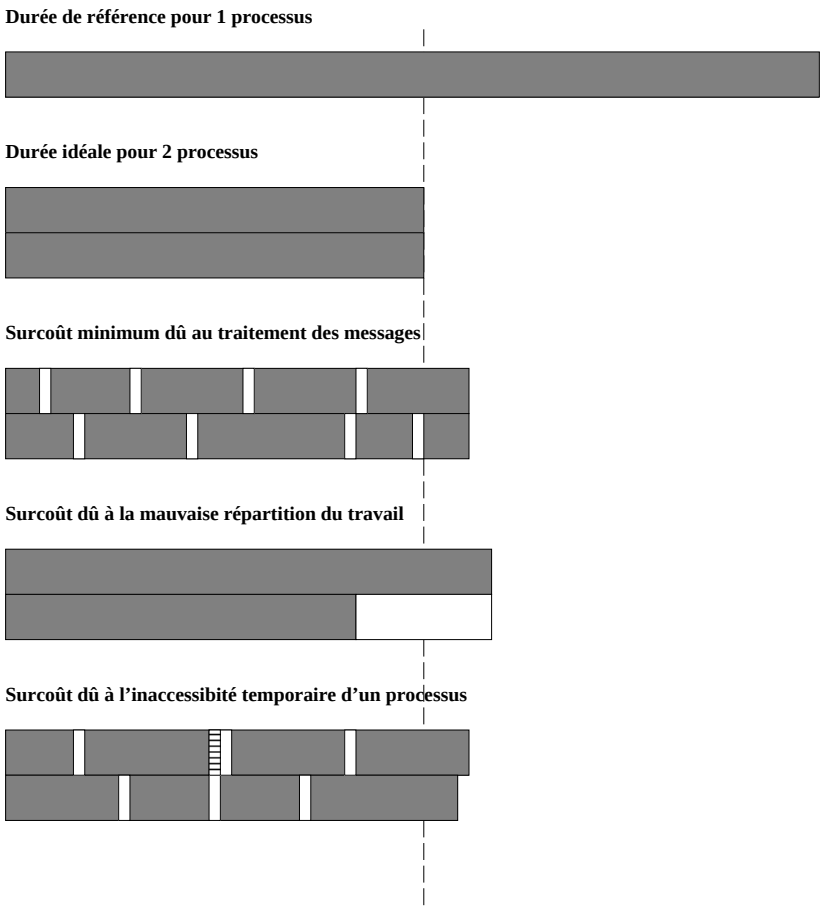

FIG. 1 – *Les différents types de surcoûts dus aux communications interprocessus*

Nous ne considérons ici que le parallélisme avec mémoire distribuée. Plusieurs processus (un par machine) s'exécutent en parallèle et peuvent communiquer par envoi de messages. Le but de la parallélisation est d'accroître linéairement les performances, c'est-à-dire d'effectuer le même travail qu'un processus unique mais en divisant son temps de résolution par le nombre de processus engagés. Pour approcher ce but, il faut principalement concilier deux objectifs antagonistes : d'une part, partager le plus équitablement la charge de travail à effectuer entre chacun des processus tout au long de la résolution pour éviter que des processus restent inactifs, d'autre part, réduire au maximum le nombre de message interprocessus afin d'éviter un surcoût en communications trop élevé.

p désigne dans cet article le nombre de processus.

3 La parallélisation du Backtrack

Nous allons rappeler brièvement les propriétés générales qui rendent efficace un algorithme parallèle de recherche en profondeur d'abord (Parallel DFS ou PDFS) de toutes les solutions. Quel que soit le type de problème considéré, lorsque l'on désire paralléliser le développement intégral de l'arbre, on suit un principe qui consiste schématiquement à répartir les racines de sous-arbres les plus proches de la racine de l'arbre de recherche entre tous les processus. Chaque processus applique une DFS à son ou ses sous-arbres parallèlement aux autres et, dès qu'il a terminé, demande un ou plusieurs nouveaux noeuds-racines à un autre processus. Dans [KGR94], il est démontré en substance que l'on peut faire en sorte que le surcoût des communications ne croisse que polynomialement en fonction du temps de résolution d'un problème tout en garantissant un partage du travail équitable. Ceci assoit de précédents résultats expérimentaux qui montraient une accélération proche de la linéarité pour des problèmes résolus par ce type d'algorithmes [FM87, FK88, RKR87]. Appliqué aux CSP, cela implique que ce surcoût devient négligeable si le problème est suffisamment grand puisque le temps de résolution est en général une fonction exponentielle du nombre de variables. Ainsi, dans [Prc96] est présenté un algorithme PDFS résolvant les CSP et dont le nombre de messages interprocessus est au pire en $O(dpn^2)$. Dans cet algorithme où les processus sont continuellement actifs jusqu'à la fin, on peut définir l'accélération par $S = \frac{T_{seq}}{T_{par}+T_{com}}$, où T_{com} est le temps total pris par les communications et T_{seq} et T_{par} les temps utilisés pour les recherches séquentielle et parallèle de solutions. Comme le nombre de nœuds développés est le même et que la charge de travail est équitablement répartie, $T_{par} = \frac{T_{seq}}{p}$. De plus, $T_{seq} = O(d^n)$ et $T_{com} = O(dpn^2)$ et donc, quand n est grand, T_{com} devient négligeable et on a $S \approx p$. Dorénavant, nous considérerons que nous avons affaire à des CSP de taille suffisante pour qu'un algorithme PDFS accélère linéairement leur temps de résolution.

4 La recherche d'une seule solution

Lorsqu'on ne désire obtenir qu'une seule solution, si on garde la même méthode en distribuant les sous-arbres le plus haut possible, la linéarité de l'accélération ne peut absolument pas être garantie. On risque de n'avoir aucune accélération ou bien d'en avoir une supralinéaire. Ceci dépend de la localisation dans l'arbre de recherche de la solution qui sera trouvée. Prenons l'exemple d'un CSP qui n'a qu'une seule solution. Le pire des cas survient quand elle se trouve dans le sous-arbre distribué le plus à gauche dans l'arbre. Tout se passe alors comme si un processus faisait la recherche en séquentiel tandis que les autres processus explorent inutilement une partie de l'arbre située après la solution. Il n'y a alors pas d'accélération. Par contre, si la solution se trouve au début d'un autre sous-arbre distribué au commencement de la résolution alors la solution est trouvée très vite et l'accélération peut être arbitrairement élevée. Si on fait bien attention à distribuer les nœuds dans le même ordre d'examen que pendant une recherche séquentielle, on peut garantir que dans le pire des cas il n'y aura pas d'accélération mais pas non plus de décélération (si on néglige le

surcoût en communication) [LW86]. Nous n'avons donc plus $T_{par} = \frac{T_{seq}}{p}$ mais uniquement $T_{par} \in]0; T_{seq}]$ soit $S \in [1; +\infty[$.

La PDFS, malgré ses défauts, a été utilisée pour plusieurs raisons : le coût en communication est faible et en moyenne l'accélération est linéaire lorsque la densité des solutions est uniforme dans tout l'arbre. L'accélération peut même être supralinéaire dans certains cas [RK93]. Cependant, lorsqu'on utilise une heuristique de choix de valeurs, on s'attend à ce que la densité des solutions soit plus forte à gauche de l'arbre, où la découverte de solutions entraîne une accélération généralement moins bonne, et on aimerait concentrer nos efforts de recherche d'abord à cet endroit. C'est pourquoi il est aussi utilisé une autre méthode consistant à développer l'arbre en partie séquentiellement en n'appliquant une recherche parallèle que sur les nœuds situés à une profondeur déterminée au fur et à mesure qu'on les trouve. Nous l'appellerons Fixed-depth PDFS ou F-PDFS. Le nombre de nœuds développés se trouve ainsi d'autant plus réduit que l'on augmente la profondeur de la distribution [Man92]. Par contre, le nombre de messages croît exponentiellement avec cette profondeur et le surcoût en communication perd son caractère polynomial. On a donc T_{com} qui croît avec la profondeur tandis que T_{par} décroît et il nous faut trouver la profondeur qui minimise la somme de ces deux termes. En pratique, le choix de la meilleure profondeur de distribution se fait expérimentalement à partir d'un échantillon de problèmes types. La profondeur de distribution est augmenté progressivement tant que l'accélération moyenne s'améliore jusqu'à ce qu'on atteigne le point où elle se met à se détériorer. Nous allons présenter une amélioration de cette méthode qui cherche à diminuer sensiblement le nombre de messages tout en gardant une variabilité de l'accélération réduite.

4.1 Un critère d'efficacité pour la distribution

Au lieu de choisir une profondeur fixe, nous allons la faire varier au cours de la recherche en fonction d'un critère qui prend en compte une évaluation heuristique du nombre de noeuds du sous-arbre à paralléliser et qui estime ainsi la taille maximum de l'espace de recherche inutilement exploré. Nous appelons cette méthode de distribution Variable-depth PDFS ou V-PDFS. On se situe, dans l'étude théorique qui suit, dans le cas où le problème possède une unique solution.

Soit $F = \frac{N_{seq}}{N_{par}}$, l'efficacité en terme d'exploration de l'espace de recherche, c'est-à-dire, le rapport entre le nombre de nœuds développés par un algorithme séquentiel DFS et le nombre de nœuds développés par un algorithme PDFS avant de trouver la solution du problème. Considérons un sous-arbre X que l'on envisage de paralléliser et désignons par N^-, le nombre de nœuds déjà développés dans l'arbre de recherche quand on parvient à la racine de X et par N^+ le nombre de nœuds qu'il faudra générer dans X avant de trouver la solution si elle s'y trouve. Nous pouvons donc exprimer ce que sera F si la solution se trouve dans X :

$$F = \frac{N_{seq}}{N_{par}} = \frac{N^- + N^+_{seq}}{N^- + N^+_{par}}$$

Soit ν, le nombre de nœuds générés par un des p processus qui s'occupent de la distribution de X (nous considérons que le travail est suffisamment bien réparti pour que chaque processus ait développé sensiblement le même nombre de nœuds à n'importe quel moment de la résolution) avant que l'un d'eux ne trouve la solution. On a $N^+_{par} = p\nu$ et $F = \frac{N^- + N^+_{seq}}{N^- + p\nu}$. Nous n'avons pas de moyen de savoir où peut se situer la solution et nous ne pouvons donc que considérer le cas le plus défavorable pour donner une borne inférieure. On sait qu'on a la relation $N^+_{seq} \geq \nu$ car nous avons vu que $T_{par} \leq T_{seq}$ lorsque son algorithme PDFS distribue les sous-arbres les plus à gauche dans l'arbre en premier. Nous obtenons donc :

$$F \geq \frac{N^- + \nu}{N^- + p\nu}$$

Comme $\frac{\partial}{\partial \nu}\left(\frac{N^- + \nu}{N^- + p\nu}\right) = \frac{(1-p)N^-}{(N^- + p\nu)^2} \leq 0$, il faut maximiser ν pour minimiser F d'où finalement:

$$F_{min} = \frac{N^- + N/p}{N^- + N}$$

où N désigne le nombre total de nœuds que contient X. F_{min} est une fonction croissante de N^- et décroissante de N donc nous pouvons d'ores et déjà déduire que plus N^- augmente (c'est-à-dire : plus on avance dans la résolution) moins N a besoin d'être petit pour garantir $F \geq F_{min}$. Il n'y a donc aucune raison valable a priori de vouloir garder la profondeur de distribution fixe tout au long de la résolution si notre but est de maintenir F au dessus d'une valeur donnée. Nous proposons donc de faire varier la profondeur de distribution des sous-arbres de telle façon qu'on ne passe pas en dessous d'un seuil d'efficacité f que l'on choisira. Ce critère de distribution est $\frac{N^- + N/p}{N^- + N} \geq f$ c'est-à-dire :

$$N \leq \frac{1-f}{f - \frac{1}{p}} N^-.$$

Si nous connaissions la valeur de N, nous pourrions déterminer si le critère est rempli pour X et si la parallélisation à partir de sa racine garantirait que $F \geq f$. Si le critère n'est pas rempli, il faut alors descendre plus profondément dans l'arbre afin de diminuer N.

Néanmoins, comme nous sommes obligés de fixer une profondeur maximale de distribution M au delà de laquelle toute parallélisation devient trop inefficace car T_{com} est trop élevé par rapport à T_{seq} et T_{par}, il y aura un certain nombre de nœuds qui seront distribués sans que le critère d'efficacité minimum ne soit rempli, notamment en début de résolution.

4.2 Evaluation heuristique du nombre de nœuds dans un sous-arbre

On ne sait pas déterminer a priori le nombre de nœuds que contient un sous-arbre sans l'avoir complètement développé dans le cas des CSP. Il devrait pouvoir être possible d'avoir une estimation plus ou moins précise de ce nombre pour certains types de problèmes particuliers où l'arbre de recherche aurait une forme régulière mais ce n'est pas envisageable dans un cadre général. Dans la mesure où nous n'avons aucune information sur la structure de l'arbre de recherche, nous proposons, dans le cas d'une recherche où le choix des variables est statique, de faire la distribution comme si tous les arbres situés à la même profondeur

possédaient le même nombre de nœuds. Ce modèle, bien que susceptible d'être fort éloigné de la réalité est pourtant celui qui est implicitement considéré lorsque la distribution est faite à une profondeur fixe : on espère qu'en moyenne la taille des sous-arbres est telle que l'on peut les distribuer à cette profondeur mais on ne tient pas compte de la variabilité de cette taille. Avec ce modèle d'arbre, nous avons la relation $N^- = kN + N_0$, où k est le nombre de sous-arbres de même profondeur que X déjà explorés et N_0 est le nombre de nœuds examinés séquentiellement avant de parvenir à X. Quand on fixe une valeur faible à M , N_0 est négligeable et $N^- \approx kN$. En introduisant cette expression dans la formule du critère, on obtient :

$$k \geq \frac{f - \frac{1}{p}}{1 - f}$$

On voit bien que k est une constante et qu'elle est indépendante de la profondeur. Le critère de distribution se trouve alors simplifié puisqu'il ne s'agit plus, quelle que soit la profondeur, que de compter le nombre de nœuds générés à cette profondeur pour savoir s'il faut paralléliser ou non. Le modèle d'arbre choisi fait que la profondeur de distribution diminuera progressivement d'unité en unité tout au long de la résolution tant qu'aucune solution n'est trouvée. On peut parler de distribution en escalier (voir la figure 2).

On peut noter qu'en choisissant $f = 100\%$, on s'assure que k soit infini et on retombe alors dans une F-PDFS. Ainsi, la V-PDFS peut être vue comme une généralisation de la F-PDFS lorsqu'on rajoute le paramètre f.

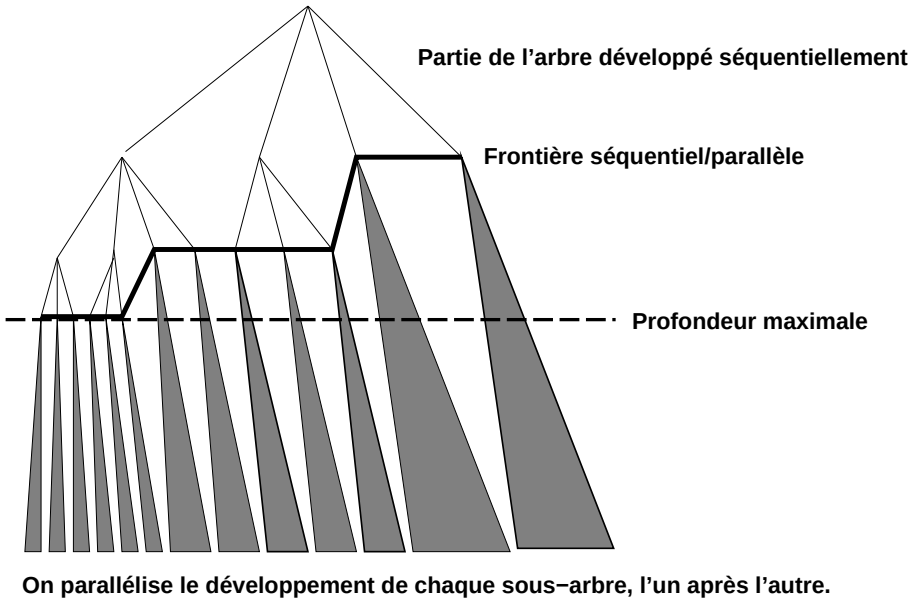

FIG. 2 – *La distribution en escalier. Les profondeurs de parallélisation des sous-arbres correspondent à $k = 2$.*

4.3 L'algorithme de recherche d'une solution

Nous pouvons concevoir un algorithme parallèle constitué de deux phases qui s'exécutent en alternance :

- DFS (séquentielle) d'un nœud qui corresponde au critère d'efficacité (c'est-à-dire qui soit au moins le $(k+1)^{eme}$ apparu à cette profondeur) ou qui soit à la profondeur maximale M (voir les lignes 4-14 de l'algorithme présenté en figure 3).

- recherche d'une solution dans le sous-arbre issu du nœud sélectionné grâce à un algorithme PDFS tel que décrit en section 3 (lignes 15-17 de l'algorithme).

```
0 Empiler la racine de l'arbre
1 Initialiser les champs de Nodes à 0
2 Prof := 0
3 TANTQUE Pile ≠ ∅ et aucune solution n'est trouvée
4     TANTQUE Pile ≠ ∅ et Prof < M et Nodes[Prof] < (f − 1/p)/(1 − f)
5         TANTQUE le sommet de la Pile ne peut générer de nœud
6             Dépiler
7             Prof := Prof - 1
8         FINTANTQUE
9         SI Pile ≠ ∅
10            Générer un nœud à partir du sommet de la Pile et l'empiler
11            Prof := Prof + 1
12            Nodes[Prof] := Nodes[Prof] + 1
13        FINSI
14    FINTANTQUE
15    SI Pile ≠ ∅
16        lancer une PDFS à partir du sommet de la Pile
17    FINSI
18 FINTANTQUE
```

FIG. 3 – *L'algorithme V-PDFS. On utilise un tableau de longueur n nommé Nodes qui compte le nombre de nœuds apparus à chaque niveau. Chaque nœud possède la prochaine variable à affecter ainsi que la liste des valeurs restantes pour produire un de ses fils. Quand on génère un nœud, on prend bien soin d'éliminer la valeur qu'on a choisi de la liste du père. Tout ceci permet de créer les nœuds un par un et de les distribuer éventuellement alors qu'ils ont déjà généré une partie de leurs fils, ce qui affine le découpage de l'arbre.*

5 Résultats expérimentaux

Nous avons choisi de faire des tests sur des CSP générés aléatoirement selon le modèle décrit dans [Pro94] sur un réseau de 8 stations SUN. Les arbres de recherche produits sont susceptibles d'être très déséquilibrés et de ne pas coller au modèle simple sur lequel nous avons basé la formule finale de notre critère. Nous avons ainsi estimé que c'était le meilleur moyen d'éprouver notre méthode de distribution et de vérifier son utilité en pratique.

Tous les tests sont faits sur deux cents instances qui contiennent toutes au moins une solution. Nous faisons varier M et f. Nous relevons le nombre total de nœuds générés et de messages interprocessus pour chaque problème afin d'évaluer l'efficacité relative de l'algorithme V-PDFS.

La figure 4 montre la distribution des valeurs F_i de l'efficacité de la recherche, ainsi que le nombre moyen de messages. Nous avons concentré notre attention sur les valeurs F_i qui sont inférieures à 100% car ce sont celles qu'on cherche à éliminer. La comparaison des qualités relatives des distributions des F_i entre elles est parfois difficile, d'autant plus qu'elle dépend de l'importance que chaque application attache à éviter les très mauvais résultats. Nous utilisons deux valeurs représentatives de l'inefficacité : $\sigma_1 = \sum_{F_i<1}(1-F_i)$ et $\sigma_2 = \sum_{F_i<1}(1-F_i)^2$.

$(f\ ,M)$	$\leq 30\%$	40%	50%	60%	70%	80%	90%	100%	$\geq 110\%$	σ_1	σ_2	msg
(100%,0)	37	9	11	12	11	7	7	10	96	54.7	36.2	78
(100%,1)	1	2	4	9	8	14	30	62	70	22.0	7.17	2000
(100%,2)	0	0	1	3	1	9	37	105	44	13.6	2.47	8400
(90%,2)	0	0	3	4	7	15	38	58	75	18.1	4.58	3700
(85%,2)	0	0	3	3	14	21	28	46	85	19.8	5.43	2800
(75%,2)	0	0	2	12	12	26	27	29	92	22.8	6.89	2300
(60%,2)	1	9	19	23	15	17	10	11	95	39.1	18.2	863
(90%,3)	0	0	2	5	7	16	35	65	70	18.3	4.39	4400
(85%,3)	0	1	2	5	10	21	30	43	88	19.0	5.11	3400
(75%,3)	0	1	1	14	9	23	34	30	88	22.7	6.76	2700

FIG. 4 – *Efficacité de la recherche et importance des communications en fonction du paramétrage (f, M) de la distribution. Les paramètres des CSP aléatoires sont un nombre de variables* $n = 16$, *une taille fixe des domaines* $d = 8$, *une densité du graphe des contraintes* $p1 = 0.5$ *et une dureté moyenne des contraintes* $p2 = 0.42$, *ce qui fait qu'en moyenne le nombre de solutions d'un problème est égal à 1.8.*

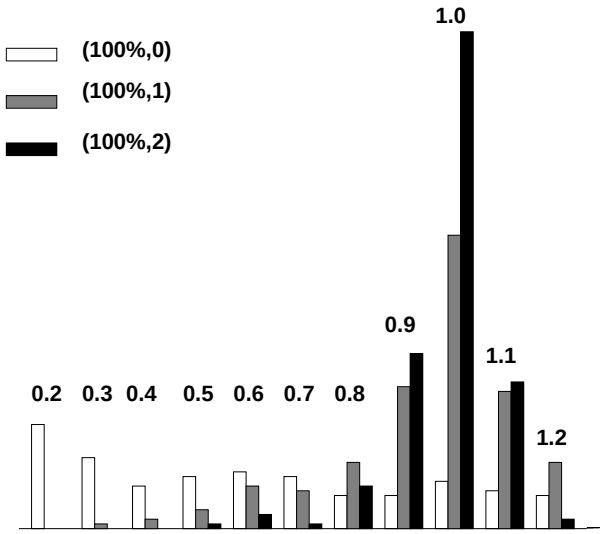

FIG. 5 – *Distributions des valeurs F_i avec une F-PDFS*

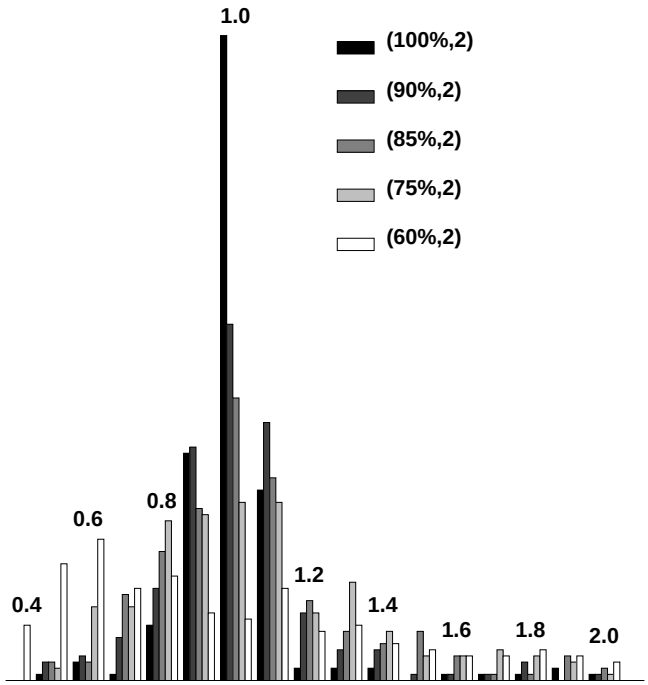

FIG. 6 – *Distributions des valeurs F_i avec une V-PDFS*

En examinant ces résultats, nous pouvons voir qu'il est difficile de déterminer si un paramétrage améliore en même temps l'efficacité de la recherche et le coût en communication par rapport à un autre. Au contraire, quand l'efficacité de la recherche est meilleure (pour des $F_i < 1$), alors les communications sont plus élevées. Plus précisément, il semble que l'efficacité de la recherche et le coût en communication soient des fonctions croissantes de f et M. Nous allons pourtant montrer que nos résultats permettent de conclure que notre méthode de distribution peut augmenter l'accélération. En effet, pour les types de problèmes susceptibles de produire un fort taux de $F_i < 1$, la qualité de l'accélération dépend du compromis que l'on fait entre F et le coût en communication. Lorsqu'on utilise un algorithme F-PDFS, nous n'avons le choix qu'entre très peu de combinaisons de valeurs qui varient de façon exponentielle (voir figures 4 et 5), tandis qu'avec un algorithme V-PDFS, on peut choisir de façon beaucoup plus fine la combinaison de valeurs qui minimise $T_{par} + T_{com}$ (voir figures 4 et 6) et qui rend par conséquent l'accélération optimale.

En pratique, la recherche du meilleur paramétrage (f, M) sera fera grâce à un échantillon représentatif de la classe du problème à paralléliser. En premier lieu, comme avec un algorithme F-PDFS, on déterminera expérimentalement la profondeur m qui maximise localement l'accélération moyenne lorsque $f = 100\%$. On sait alors que ($f = 100\%$, $m - 1$) et ($f = 100\%$, $m + 1$) produisent une moins bonne accélération que ($f = 100\%$, m). En fixant M à $m + 1$ et f à 100% nous sommes sûr d'avoir dépassé le point où l'accélération est la meilleure. Il ne nous reste plus qu'a faire diminuer f pour nous rapprocher du point (f, M) qui maximise l'accélération.

6 Conclusion et perspectives

Nous avons présenté l'algorithme V-PDFS qui permet de sélectionner dynamiquement la profondeur la plus adéquate pour commencer à distribuer les nœuds. En rajoutant un nouveau paramètre, nous nous sommes donné un nouveau moyen d'affiner la distribution et d'améliorer l'accélération résultante.

Par ailleurs, nous espérons pouvoir étendre nos résultats aux problèmes d'optimisation résolus par des méthodes telles que le Branch and Bound, qui gère une borne de coût de la solution localement la meilleure et l'utilise pour élaguer des branches de l'arbre en cours de résolution. Il est fréquent que leur parallélisation entraîne une augmentation de nœuds générés parce qu'un processus peut commencer à développer une partie de l'arbre qui aurait été élaguée en séquentiel [Rou89]. On peut considérer que la recherche d'une seule solution équivaut en optimisation à la découverte d'une nouvelle borne qui permet d'élaguer totalement le reste de l'arbre. Comme cette situation extrême est celle qui est la plus susceptible de faire perdre de l'efficacité à la parallélisation d'un problème d'optimisation, la formule de F lui est valable. Nous pourrions ainsi appliquer directement notre méthode pour la recherche de la meilleure solution, tout en espérant que pour un même coût en communication on puisse avoir une meilleure efficacité de recherche que pour l'obtention d'une seule solution.

Références

[FK88] C. Fergusson and R.E. Korf. Distributed Tree Search and its Application to Alpha-Beta Pruning. In *AAAI'88*, pages 128–132, 1988.

[FM87] R. Finkel and U. Manber. DIB—A Distributed Implementation of Backtracking. *ACM Transaction on Programming Languages and Systems*, 9(2):235–256, 1987.

[KGR94] V. Kumar, A. Grama, and V. Rao. Scalable load balancing techniques for parallel computers. *Journal of Parallel and Distributed Computing*, 22(1), 1994.

[LW86] G.-J. Li and B.W. Wah. How Good are Parallel and Ordered Depth-First Searches. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 992–999. IEEE Computer Society Press, 1986.

[Man92] B. Mans. *Contribution à l'algorithmique non numérique parallèle : parallèlisations de méthodes de recherche arborescentes.* Thèse de doctorat, Université Paris VI, pages 169-182, 1992.

[Prc96] N. Prcovic. A Distributed Algorithm Solving CSP s with a Low Communication Cost. In *Proceedings of the* 8^{th} *International Conference on Tools with Artificial Intelligence*, 1996.

[Pro94] P. Prosser. Binary constraint satifaction problems : Some are harder than others. In *11th European Conference on Artificial Intelligence*, pages 95–99, 1994.

[RK93] V. N. Rao and V. Kumar. On the Efficiency of Parallel Backtracking. In *IEEE Transactions on Parallel and Distributed Systems*, volume 4:4, pages 427–437, 1993.

[RKR87] V.M. Rao, V. Kumar, and K. Ramesh. A Parallel Implementation of Iterative-Deepening-A*. In *AAAI'87*, pages 178–182, 1987.

[Rou89] C. Roucairol. Parallel branch and bound algorithms, an overview. Rapport de recherche 962, INRIA Rocquencourt, 1989.