L'élargissement aléatoire progressif

Tristan Cazenave
LIASD - Université Paris 8
2, rue de la liberté
93520 Saint-Denis
cazenave@ai.univ-paris8.fr

Résumé

Nous présentons un algorithme qui permet de résoudre efficacement les problèmes de complétion de quasi-groupes. Il se greffe sur un algorithme de Forward-Checking. Il élargit progressivement le nombre de valeurs à explorer. Pour un nombre donné de valeurs à explorer, et pour chaque variable, il choisit aléatoirement entre le nombre de valeurs à explorer et ce nombre moins un. La probabilité de choisir le nombre de valeurs à explorer est elle aussi augmentée progressivement. Sur 9900 problèmes de complétion de quasi-groupes d'ordre 20 et pour une bonne heuristique d'ordonnancement des valeurs utilisant une modélisation duale, notre algorithme donne de meilleurs résultats que le Forward-Checking, que les recommencements rapides aléatoires (Rapid Randomized Restarts), que l'élargissement itératif (Iterative Broadening) et que la recherche avec déviations limitées (Limited Discrepancy Search).

Abstract

We present an algorithm that solves quasi-group completion problems. It consists in a slight modification of the Forward-Checking algorithm. It progressively increases the number of values to explore. For a given number of values, and for each variable, it randomly chooses between the number of values to explore and this number minus one. The probability of choosing the number of values to explore is also progressively increased. On 9,900 quasi-group completion problems of order 20, and for a good value ordering heuristic derived from two dual models, our algorithm gives better results than Forward-Checking, Rapid Randomized Restarts, Iterative Broadening, and Limited Discrepancy Search.

1 Introduction

Nous nous intéressons aux algorithmes qui élargissent progressivement le nombre de choix autorisés de façon à trouver aussi vite que possible une solution aux problèmes qu'ils traitent. Notre algorithme est une modification assez simple de l'algorithme de Forward-Checking ; elle consiste à choisir aléatoirement les variables pour lesquelles nous autorisons un choix de plus que les autres. C'est en ce sens un algorithme plus progressif que l'élargissement itératif [2].

Nous le comparons à d'autres algorithmes qui permettent de résoudre le problème de la complétion de quasi-groupes, à savoir le Forward-Checking, la recherche avec déviations limitées (LDS) [5], l'élargissement itératif [2], et les recommencements rapides aléatoires (RRR) [4].

La deuxième section présente le problème de la complétion de quasi-groupes, la troisième section expose différents algorithmes qui peuvent être utilisés pour le résoudre, la quatrième section présente l'élargissement aléatoire progressif, la cinquième section donne les résultats expérimentaux, la sixième section est la conclusion.

2 Le problème de la complétion de quasi-groupes

Nous commençons par décrire le problème de la complétion de quasi-groupes. Nous détaillons ensuite comment nous avons engendré des problèmes. Nous rappelons enfin comment une modélisation redondante permet d'accélérer la résolution.

2.1 Description du problème

Un quasi-groupe est un carré latin $n \times n$. C'est une matrice dont chaque ligne et chaque colonne contient n éléments différents. Il n'y a que n valeurs différentes possibles.

Le problème de la complétion de quasi-groupes

TAB. 1 – Un quasi-groupe d'ordre 4.

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

consiste à compléter un quasi-groupe comportant des trous. C'est un problème intéressant pour tester des algorithmes de satisfaction de contraintes car il est structuré. De plus, il est aisé d'engendrer des problèmes arbitrairement difficiles. C'est un problème qui a une transition de phase ; les problèmes ayant peu de trous sont faciles, puis les problèmes deviennent plus difficiles quand on augmente le nombre de trous, enfin les problèmes ayant beaucoup de trous redeviennent faciles. La difficulté d'un problème est liée au nombre de trous qu'il contient.

Le problème de la complétion de quasi-groupes a la propriété d'avoir une répartition des temps de résolution par un algorithme de backtrack qui a une *heavy-tail* [3]. Pour éliminer cette propriété la stratégie des recommencements rapides aléatoires a été proposée [4].

2.2 Création des problèmes

Pour chaque problème, nous avons engendré un quasi-groupe complet à partir d'une matrice vide à l'aide des recommencements rapides aléatoires. L'ordonnancement des valeurs est dans ce cas aléatoire. Puis, pour chaque problème, nous avons enlevé aléatoirement autant de valeurs qu'il fallait pour atteindre le pourcentage de trous voulu. Pour chaque pourcentage de trous allant de 1 à 99, nous avons engendré 100 problèmes. Soit un total de 9900 problèmes.

2.3 La modélisation redondante

La modélisation redondante a déjà été utilisée avec succès pour le problème de la complétion de quasi-groupes [1]. Le modèle primaire pour un quasi-groupe d'ordre n est d'utiliser n^2 variables, une pour chaque élément de la matrice, le domaine de valeurs pour toutes les variables étant $D = \{k | 1 \leq k \leq n\}$. On nomme les variables x_{ij}, i correspondant à la ligne et j à la colonne de la variable. Les n^2 contraintes pour les lignes sont $x_{ij} \neq x_{il}$ avec $j \neq l$, et les n^2 contraintes pour les colonnes sont $x_{ij} \neq x_{lj}$ avec $i \neq l$. Le modèle dual pour les lignes consiste à considérer dans quelle colonne d'une ligne donnée se trouve une valeur donnée. On nomme les variables duales r_{ik}, i correspondant à la ligne et k à la valeur. On obtient ainsi n^2 contraintes de la forme $r_{ik} \neq r_{il}$ avec $l \neq k$, et n^2 contraintes de la forme $r_{ik} \neq r_{jk}$ avec $i \neq j$. Le modèle dual pour les colonnes consiste à considérer dans quelle ligne d'une colonne donnée se trouve une valeur donnée. On nomme les variables duales c_{jk}, j correspondant à la colonne et k à la valeur. On obtient alors n^2 contraintes de la forme $c_{jk} \neq c_{jl}$ avec $k \neq l$, et n^2 contraintes de la forme $c_{jk} \neq c_{lk}$ avec $j \neq l$. Les contraintes qui lient les modèles entre eux sont : $x_{ij} = k \Leftrightarrow r_{ik} = j$ pour les lignes, et : $x_{ij} = k \Leftrightarrow c_{jk} = i$ pour les colonnes. C'est exactement la modélisation redondante décrite dans [1]. Une bonne heuristique d'ordonnancement des valeurs qui permet d'accélérer la résolution des problèmes à l'aide de la modélisation redondante est *l'heuristique de la somme des domaines* (vdom+). Elle consiste à choisir en priorité les valeurs dont les deux variables duales ont des tailles de domaines dont la somme est minimale. On peut noter que le Forward-Checking avec cette modélisation redondante est presque équivalent à la consistance d'arcs, à l'exception de l'ordre d'instanciation des variables singleton.

3 Algorithmes de recherche

Les algorithmes de recherche qui sont liés au problème de la complétion de quasi-groupes sont le Forward-Checking, les recommencements rapides aléatoires, l'élargissement itératif, et la recherche avec déviations limitées.

3.1 Les recommencements rapides aléatoires

Les recommencements rapides aléatoires [4] consistent à recommencer une recherche après un nombre fixé de backtracks. Ils utilisent des choix aléatoires dans la recherche pour varier les essais. Par exemple lorsque plusieurs variables ont le nombre minimum de valeurs, l'algorithme choisit aléatoirement entre ces variables.

L'intérêt des recommencements rapides aléatoires pour la complétion de quasi-groupes est qu'ils éliminent le caractère *heavy-tail* de la recherche. En effet, si on résout plusieurs fois de suite le même problème, l'algorithme de Forward-Checking peut avoir des temps de calculs très longs ou très rapides sur ce même problème. Les recommencements rapides aléatoires utilise cette propriété faisant de nombreuses recherches Forward-Checking très courtes. Ils donnent de bien meilleurs résultats que le Forward-Checking pour la complétion de quasi-groupes. De plus ils éliminent la répartition *heavy-tail* des temps de résolution de problèmes.

3.2 L'élargissement itératif

L'élargissement itératif [2] consiste à n'envisager qu'un nombre limité de choix à chaque étape de la recherche. Pour un algorithme de satisfaction de contraintes, cela revient à avoir un seuil n, et à ne backtracker que sur les n premières valeurs de chaque variable. L'élargissement itératif commence par chercher sur les deux premières valeurs de chaque variable, puis sur les trois premières valeurs, et ainsi de suite jusqu'à trouver une solution.

Pour notre problème l'élargissement itératif n'est pas très intéressant car lorsqu'on passe de une à deux valeurs, la complexité de la recherche croit très rapidement à cause du grand nombre de variables du problème. Nous l'avons cependant testé essentiellement pour le comparer avec l'élargissement aléatoire progressif qui effectue aussi un élargissement itératif mais plus progressif.

3.3 La recherche avec déviations limitées

La recherche avec déviations limitées (LDS) est due à Harvey et Ginsberg [5]. Une déviation est un noeud dans l'arbre de recherche pour lequel l'algorithme ne suit pas l'heuristique. Pour une feuille de l'arbre de recherche, son ordre est le nombre de déviations qui sont nécessaires pour l'atteindre. LDS explore les feuilles en ordre croissant.

LDS diversifie bien l'exploration de l'espace de recherche. Lorsqu'on dispose d'une bonne heuristique d'ordonnancement des valeurs qui ne se trompe qu'un petit nombre de fois, LDS donne de bons résultats.

4 L'élargissement aléatoire progressif

L'élargissement aléatoire progressif (PRB) est inspiré de l'élargissement itératif puisqu'il augmente aussi la taille des arbres qu'il recherche à chacune de ses étapes. Il le fait simplement de manière plus progressive : contrairement à l'élargissement itératif il n'augmente pas d'un coup le nombre de valeurs à envisager pour toutes les variables, mais fixe à chaque étape une probabilité que les variables soit élargies.

Chaque étape de la recherche est paramétrée par deux valeurs :

– Son *ordre*, qui est le nombre maximum de valeurs qui peuvent être envisagées à chaque noeud.
– Sa *proba*, qui est la probabilité pour une variable d'avoir *ordre* valeurs explorées.

En utilisant une fonction *rand*() qui renvoie un nombre pseudo-aléatoire entre 0 et 1, la fonction principale de l'algorithme est la suivante :

```
bool prand (ordre, proba) {
  if (timeout ())
    return false;
  d = chooseVariable ();
  if (d == NULL)
    return true;
  if (rand () > proba))
    ordrelocal = ordre - 1;
  else
    ordrelocal = ordre;
  for (k = 0; ((k < ordrelocal) &&
               (k < nb_values)); k++) {
    valeur = chooseValue (k);
    affectValue (d, valeur);
    if (consistent (d, valeur))
      if (prand (ordre, proba))
        return true;
    restoreVariable ();
    undoAffectValue (d, valeur);
  }
  return false;
}
```

Dans cette fonction, *ordre* est le nombre maximum de valeurs qui peuvent être considérées pour une variable, et *proba* est la probabilité pour une variable d'avoir *ordre* valeurs explorées.

La fonction *prand* correspond à une étape de l'élargissement aléatoire progressif. Une fonction appelante qui augmente l'ordre et la probabilité, est aussi nécessaire. La boucle la plus interne augmente la probabilité, alors que la boucle externe augmente l'ordre :

```
bool prb () {
  for (ordre = 2; ordre <= size; ordre++) {
    for (p = 0.05; p <= 1.0; p += 0.05) {
      if (prand (ordre, p))
        return true;
      if (timeout ())
        return false;
    }
  }
  return false;
}
```

L'élargissement aléatoire progressif est plus graduel que l'élargissement itératif. De plus, il effectue des choix aléatoires, ce qui lui permet à la fois de tirer partie de la répartition très variable des temps de résolution, et de bien répartir son effort sur toutes les variables.

TAB. 2 – Temps pour différentes versions des recommencements rapides aléatoires avec un timeout de 1000 secondes et un ordre aléatoire des valeurs.

%	rrr(6)	rrr(7)	rrr(8)
35%	0.49	0.57	0.72
36%	1.60	2.17	1.13
37%	3.74	6.51	4.12
38%	17.35	13.06	18.57
39%	30.49	31.05	26.48
40%	114.73	74.96	309.14
41%	90.78	79.49	67.37
42%	552.72	421.25	438.58
43%	140.42	108.89	121.22
44%	168.26	87.31	160.46
45%	290.77	198.11	143.13
46%	58.00	49.03	54.86
47%	46.12	45.97	42.36
48%	43.82	39.28	33.79
49%	38.07	22.10	27.79
50%	10.18	15.68	11.44
51%	11.54	10.21	8.91
52%	9.60	7.04	5.77
53%	5.87	5.56	5.15
54%	4.51	4.59	4.19
55%	3.83	3.42	3.22
total	1675.26	1256.69	1517.21

TAB. 3 – Temps pour différentes versions des recommencements rapides aléatoires avec un timeout de 1000 secondes et un ordre des valeurs utilisant les modèles duaux.

%	rrr(6)	rrr(7)	rrr(8)
35%	0.39	0.49	0.26
36%	0.65	0.86	0.65
37%	1.47	2.39	3.24
38%	6.48	4.65	5.69
39%	9.44	6.50	6.74
40%	211.03	39.33	166.11
41%	57.86	19.02	17.90
42%	37.33	33.14	42.88
43%	13.83	16.38	16.72
44%	22.07	11.25	11.88
45%	10.82	7.93	6.15
46%	5.34	3.16	8.39
47%	5.17	3.95	4.05
48%	2.64	2.30	2.02
49%	1.85	2.47	1.48
50%	1.33	1.16	1.25
51%	1.12	1.04	0.92
52%	>1000.93	>1000.77	>1000.81
53%	1.08	0.82	0.63
54%	0.65	0.55	0.56
55%	0.64	0.57	0.55
total	>1409.60	>1175.30	>1315.13

5 Résultats expérimentaux

La machine utilisée pour tester les algorithmes est un Pentium D 2,8 GHz. Nous avons utilisé pour tous les algorithmes l'heuristique du plus petit domaine comme heuristique de choix de la variable. Lorsque plusieurs variables ont un plus petit domaine de même taille, les algorithmes effectuent un choix aléatoire entre toutes ces variables.

Pour chaque pourcentage de trous, les algorithmes sont testés sur les 100 problèmes créés, soit sur un total de 9900 problèmes.

Nous avons d'abord testé les recommencements rapides aléatoires avec une heuristique aléatoire de choix des valeurs, de façon à établir des résultats de référence pour nos problèmes qui correspondent aux algorithmes déjà publiés pour le problème. Nous avons testé les valeurs du seuil pour un nombre de backtracks allant de 1 à 15 ; les meilleurs résultats ont été trouvés pour 7 backtracks. Les résultats pour les seuils 6, 7 et 8 sont donnés dans le tableau 2 ; le total correspond au temps de résolution des 9900 problèmes.

Dans les expériences suivantes, tous les algorithmes utilisent l'heuristique de somme minimale des domaines duaux comme heuristique d'ordonnancement des valeurs. Lorsque plusieurs valeurs ont une même valeur heuristique, les algorithmes procèdent à un choix aléatoire entre ces valeurs.

Nous avons testé différents nombres de backtracks pour les recommencements rapides aléatoires (de 1 à 15) et les meilleurs résultats ont été trouvés pour 7 backtracks. Le tableau 3 donne pour les pourcentages les plus difficiles, le temps mis pour résoudre les cent problèmes créés. Seules les valeurs pour 6, 7 et 8 backtracks sont détaillées. Un premier résultat est que l'ordonnancement des valeurs profite aux recommencements rapides aléatoires comme on peut le voir en comparant aux résultats du tableau 2.

Le tableau 4 donne les mêmes informations pour les mêmes problèmes pour les algorithmes de Forward-Checking (fc), d'élargissement itératif (ib), de recherche avec déviations limitées (lds), et d'élargissement aléatoire progressif (prb).

Même si l'on ôte le problème difficile que les recommencements rapides aléatoires n'arrivent pas à résoudre pour 52% de trous, l'élargissement aléatoire progressif prend moins de 110 secondes alors que les recommencements rapides aléatoires prennent 175

Tab. 4 – Temps pour différents algorithmes avec un timeout de 1000 secondes.

%	fc	ib	lds	prb
35%	0.28	0.21	0.46	0.38
36%	0.36	0.45	0.76	0.62
37%	2.19	1.57	1.29	1.53
38%	3.04	4.13	3.73	4.11
39%	11.26	6.07	8.60	4.74
40%	23.39	70.01	22.30	17.15
41%	104.02	46.70	16.96	8.29
42%	>2 437.46	>1 737.28	47.84	17.28
43%	>2 364.21	>3 560.11	27.13	11.38
44%	>4 041.71	>2 556.47	30.09	8.68
45%	>5 605.75	>5 317.20	25.54	5.32
46%	>8 197.77	>7 848.10	13.64	4.09
47%	>9 333.54	>11 409.51	14.24	3.13
48%	>8 460.48	>7 927.31	10.05	2.48
49%	>13 806.12	>8 601.95	8.80	1.54
50%	>11 728.74	>10 501.10	6.09	1.62
51%	>7 987.08	>9 353.75	5.91	1.24
52%	>7 309.98	>12 359.65	4.75	1.34
53%	>8 032.81	>11 070.98	3.55	0.80
54%	>3 969.22	>7 897.92	2.99	0.61
55%	>5 358.35	>7 590.85	2.72	0.54
total	>160 940	>163 241	289.07	109.41

secondes. L'élargissement aléatoire progressif donne donc de meilleurs résultats que les recommencements rapides aléatoires.

On peut aussi remarquer que les problèmes les plus difficiles pour RRR, LDS et PRB se trouvent entre 40% et 44% de trous, alors que les problèmes les plus difficiles pour le Forward-Checking se trouvent plutôt entre 46% et 50%, ces derniers étant facilement résolus par RRR, LDS et PRB.

La figure 1 donne la répartition des temps mis par RRR(7) et PRB pour les problèmes les plus difficiles pour ces algorithmes, c'est à dire avec 42% de trous. Les problèmes sont triés par temps de résolution croissant. On voit bien que PRB se comporte mieux que RRR pour les problèmes difficiles.

La figure 2 donne une répartition construite de la même façon pour le Forward-Checking et les problèmes comportant 49% de trous. On voit que la répartition ne suit pas la même courbe que RRR(7) ou que PRB. La courbe pour le Forward-Checking correspond à une distribution *heavy-tail*.

6 Conclusion

Le résultat principal est que l'élargissement aléatoire progressif donne de meilleurs résultats pour la complétion de quasi-groupes d'ordre 20 que les autres algorithmes qui lui sont habituellement associés, à savoir le Forward-Checking et les recommencements rapides aléatoires. Nous avons aussi montré que l'élargissement aléatoire progressif est plus rapide que l'élargissement itératif et que la recherche avec déviations limitées. Un autre résultat est que la modélisation redondante est une alternative intéressante à l'ordonnancement aléatoire des valeurs pour les recommencements rapides aléatoires.

Ces bons résultats sont sans doute dus aux caractéristiques de l'élargissement aléatoire progressif : il fait croître l'espace de recherche plus graduellement que l'élargissement itératif, il utilise l'aléatoire à chaque étape de sa recherche ce qui lui permet de tirer partie de la distribution *heavy-tail* des temps de résolutions, et il diversifie sa recherche sur les différentes variables mais d'une manière différente de LDS.

Dans des travaux futurs, il serait intéressant de faire dépendre l'élargissement d'une heuristique plutôt que d'un choix aléatoire, et de l'appliquer à d'autres problèmes combinatoires.

Références

[1] Ivan Dotu, Alvaro del Val, and Manuel Cebrian. Redundant modeling for the quasigroup completion problem. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2003.

[2] M. L. Ginsberg and W. D. Harvey. Iterative broadening. *Artificial Intelligence*, 55(2-3) :367–383, 1992.

[3] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24 :67–100, 2000.

[4] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *AAAI-98*, pages 431–437, 1998.

[5] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In Chris S. Mellish, editor, *IJCAI-95*, pages 607–615, Montréal, Québec, Canada, August 20-25 1995. Morgan Kaufmann.

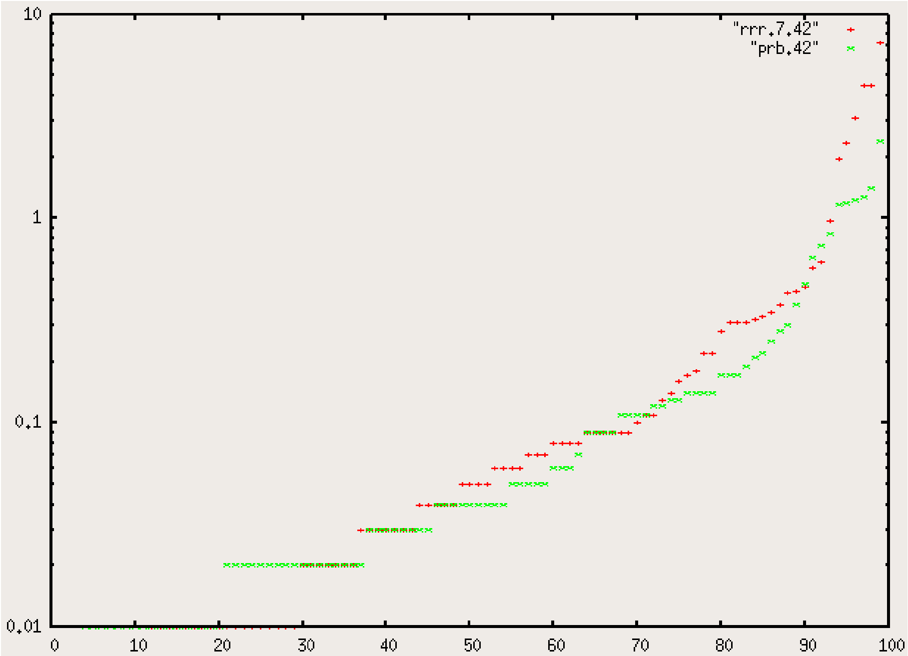

FIG. 1 – Comparaison de rrr(7) et prb pour 42% de trous.

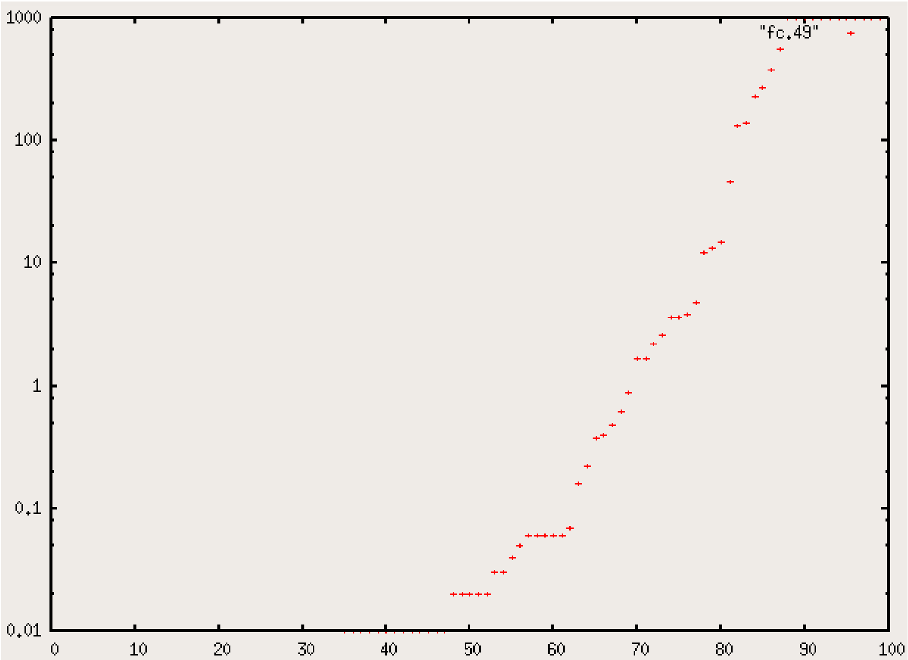

FIG. 2 – Répartition des temps du Forward-Checking pour 49% de trous.