AMU – L3 Informatique – 2016-2017 – Montperrin – IA

Futoshiki

Rapport de projet

Thomas MEDARD

Kurt SAVIO

Enseignant : Cyril TERRIOUX

Table des matières

Utilisation du programme

Notre programme contient un Makefile pour être compilé.

Pour ce faire entrez

```
make all
```

Une fois compilé, un fichier exécutable devrait être crée. Celui-ci se nomme "futoshiki" (ou futoshiki.exe si vous compilez sous Windows).

Pour exécuter le programme tapez une commande ayant ce profile :

```
futoshiki [-b] [-f] [-fh] [-hn (0 < n < 4)] cheminVersLaGrille
```

Avec "-b" correspondant à l'algorithme backtrack, "-f" à l'algorithme forward checking et "-fh" à l'algorithme forward checking avec des heuristiques (voire section « heuristiques »).

Si vous décider d'utiliser « -fh », vous pouvez préciser « -hn » avec n un nombre entre 1 et 3 pour désactiver une heuristique (elles sont toutes utilisées par défaut) (voire section « heuristiques » pour les détailles).

- Pour n = 1 : désactiver l'heuristique du plus petit domaine valide.
- Pour n = 2 : désactiver l'heuristique des contraintes autres que DIF.
- Pour n = 3 : désactiver l'heuristique de fréquence des valeurs.

Ainsi mettre « -h1 -h2 -h3 » revient à utiliser « -f ».

La grille doit être formée de la manière suivante :

Chaque fichier contient une grille carrée.

La première ligne contient la dimension n de la grille (la grille est alors de taille n × n).

Les 2n − 1 lignes suivantes sont une alternance des lignes suivantes :

- Les lignes indiquant les indices horizontaux : chaque case de la grille est matérialisée par un chiffre. Il s'agit d'un zéro si la case est initialement vide, d'un chiffre entre 1 et n si la case a initialement une valeur fixée. Deux cases voisines sont séparées par un espace s'il n'y a pas d'indice entre elle, par un < ou un > sinon.
- Les lignes indiquant les indices verticaux : un point entre deux cases voisines de la même colonne matérialise l'absence d'indice tandis qu'un accent circonflexe ou la lettre « v » symbolise l'existence d'un indice.

Il est à noter que pour des grilles particulièrement compliquées, certains algorithmes peuvent prendre du temps pour la résolution ou pour arriver à la conclusion qu'il n'y a pas de solution.

Structures de données

Toutes les structures de données sont définis dans le fichier « define.h ».

Il est aussi à noter que les valeurs courantes ainsi que celles du domaine ne sont jamais supprimées mais plutôt remplacées par NO_DOMAINE (désolé pour le franglais).

Représentation de la grille

La grille est représentée par deux type de structure.

- CASE qui représente une variable. Elle contient l'indice de la case, un tableau des valeurs de son domaine de base, une liste d'adjacence des contraintes auxquelles elle est liée (voir plus loin) ainsi que l'indice de la dernière case du tableau précédent et enfin un lien vers son affectation (voir représentation CSP).
- CONTRAINTE qui représente une contrainte. Elle contient l'opérateur (DIF, SUP, INF), un lien vers la case opérande de gauche et un lien vers la case opérande de droite.

Les cases sont rangées dans un tableau de dimension n * n qui représente la grille. Les contraintes sont rangées dans un tableau linéaire. Ces deux tableaux ainsi que leur taille sont passés au différentes fonctions pour qu'elles puissent y accéder.

Représentation CSP

Le CSP est représenté à l'aide d'une structure principale et d'une secondaire spécifique à forward checking.

- AFFECTATION représente une variable que l'on peut traitée dans le cadre de la résolution. En effet les algorithmes ne modifient jamais directement les CASE pour éviter d'altérer la grille de base. Ces affectations possèdent un lien vers la case correspondante, le domaine courant de l'affectation (potentiellement modifié au fur et à mesure de la résolution), l'éventuelle valeur courante, et deux variables spécifiques à forward checking :
  - compt qui représente le nombre de domaine modifier par l'affectation courante
  - previousDomain qui représente le domaine courant avant de commencer à tester une affectation.
- OLD_DOM, qui est spécifique à forward checking, représente un changement de domaine lors d'une affectation. Il contient les anciennes valeurs du domaine avant la modification, l'indice de cette structure dans le sous tableau (voir plus loin), le lien vers l'affectation qui a effectué la modification.

Les affectations sont stockés dans un tableau qui est propre à chaque algorithme. Ainsi ceux-ci sont indépendant car ce sont eux qui s'occupent de les initialiser et de les détruire à la fin de la résolution.

Les structures OLD_DOM sont stockées dans une matrice dont les indices du premier niveau correspond aux indices des variables, le deuxième niveau correspond à chaque OLD_DOM de cette variable, c'est-à-dire toutes les modifications de son domaine effectuées par d'autres affectations.

Heuristiques

Choix de variable

Variable avec plus petit domaine valide

Cette heuristique consiste à prendre en priorité les variables avec le plus petit domaine. De cette façon, on commencera par les variables qui ont une contrainte de valeur (celles qui ne peuvent prendre qu'une seul valeur) puis par celles qui ont été le plus modifiées. De plus plus, de cette façon on remontera plus vite en cas d'échec provenant d'affectation antérieur car il y aura moins de valeur à tester.

Pour réaliser cette heuristique, on créer un tableau de pointeur vers toutes les AFFECTATION. Ensuite on applique un tri quicksort en fonction des tailles des domaines.

Ce tri pourrait être appliqué à chaque fois que l'on a une affectation valide (et donc que l'on a effectivement modifié des domaines) mais un problème avec les pointeurs fait que sur certaines grilles, l'algorithme est faux.

Pour cohabiter avec les autres heuristiques de choix de variable, on applique cette heuristique jusqu'aux variables avec un domaine valide de taille n / 2, puis si les autres heuristiques n'ont pas sélectionné une variable, alors on reprend la recherche par taille de domaine.

L'avantage de cette heuristique est que l'on sélectionne toujours une variable qui permettra de remonter rapidement. L'inconvénient est que l'on doit appliquer le quicksort.

Variable avec contrainte autre que DIF

Cette heuristique consiste à prendre les variables avec des contraintes autre que différence (SUP ou INF). Ainsi la variable fera plus de modifications et pourra faire un échec plus rapidement si on a effectué une mauvaise affectation antérieure.

Pour réaliser cette heuristique, on créer un tableau de pointeur que l'on remplis lors de l'initialisation avec des pointeurs vers toutes les variables avec une contrainte autre que de différence.

L'avantage est que l'on remontera plus rapidement en cas de mauvais affectation. L'inconvénient est qu'il faut créer un tableau de pointeurs (ce qui n'est pas un grand inconvénient car cela est fait uniquement lors de l'initialisation).

Choix de valeur

Selon la fréquence des valeurs

Cette heuristique consiste à prendre la valeur valide la moins utilisée. Ainsi il y a plus de chance que celle-ci soit juste notamment par rapport aux contraintes de différences sur la ligne et la colonne.

L'avantage est que la valeur choisit à plus de chance d'être valide. L'inconvénient est qu'il faut maintenir à jour un tableau des fréquences et qu'à chaque choix de valeur, il faut parcourir toutes les valeurs possibles pour prendre la moins fréquente.

Comparaison des résultats

Les tests ont été effectués sur la machine suivante :

Processor : AMD FX(tm)-4300 Quad-Core Processor (3.80 Ghz)

Memory : 8152MB(667)

Windows Version : Microsoft Windows 10 Famille (x64)

Disk Drive : WDC WD10 EZEX-00KUWA0 SATA Disk Device(931GB,IDE)

Il faut noter que l'environnement comprend d'autres programmes fonctionnant en même temps que ces tests.

Il faut aussi noter que nous n'avons pas inclus les grilles en 9x9 car celles-ci prennent trop de temps à être résolues (et ce même avec les heuristiques) (nous avons déjà essayé de laisser le programme s'exécuter pendant plus de 8 heures).

Les grilles 1 sont des grilles « simples »(peu de contraintes de supériorité ou infériorité et pas de contrainte de nombre déjà placés), tandis que les grilles 2 sont des grilles « complexes ».

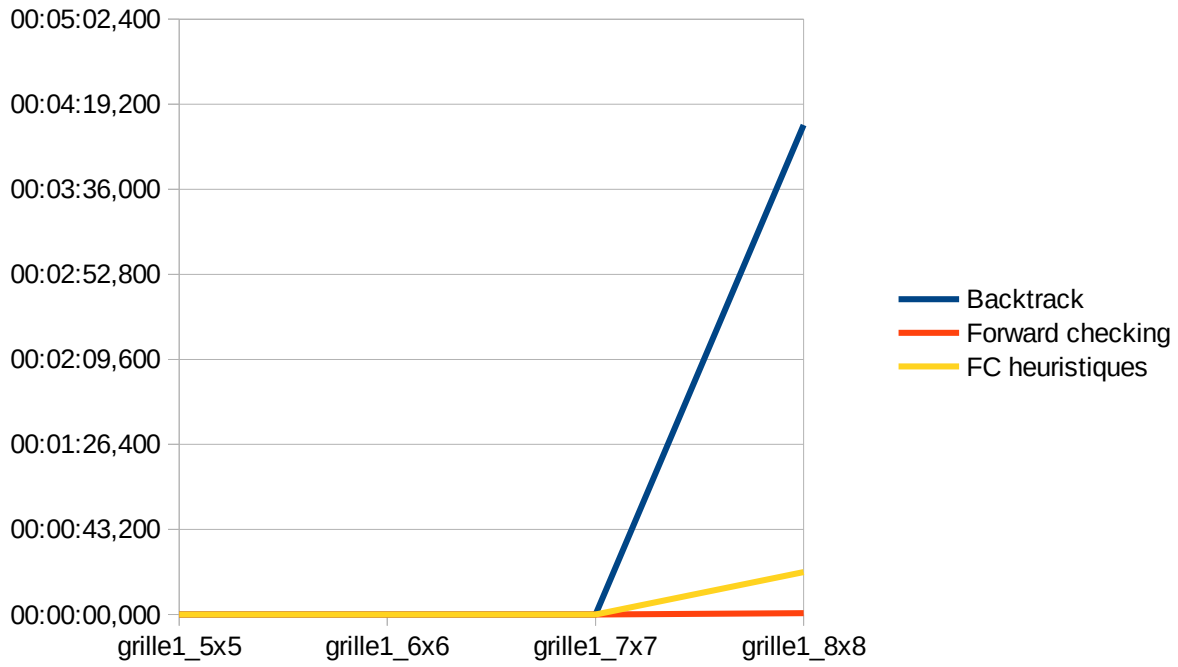
Toutes les courbes ont été réalisées avec les données brutes présentes dans le tableau suivant.

Grilles	Résultats	Algorithmes								
		Backtrack	Forward checking	Forward checking heuristiques	FC -h1	FC -h2	FC -h3	FC -h1 -h2	FC -h1 -h3	FC -h2 -h3
grille1_5x5.fut	Temps	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000
	Nombre de noeuds	4450	789	275	115	789	275	789	115	789
	Nombre de tests de contraintes	74072	9091	3233	1427	9091	3231	9091	1427	9091
grille1_6x6.fut	Temps	00:00:00,015	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,015	00:00:00,000	00:00:00,000
	Nombre de noeuds	55459	9297	1202	294	2143	1370	9297	296	33040
	Nombre de tests de contraintes	1172979	130085	19508	4467	29573	21780	130085	4499	33040
grille1_7x7.fut	Temps	00:00:00,025	00:00:00,015	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,015	00:00:00,000	00:00:00,000
	Nombre de noeuds	524822	20259	3700	1080	7165	2699	20259	1339	95101
	Nombre de tests de contraintes	13379420	359124	54811	16533	110321	38862	359124	20202	95101
grille1_8x8.fut	Temps	00:04:08,610	00:00:00,703	00:00:21,515	00:00:09,453	00:00:14,531	00:00:19,828	00:00:00,968	00:00:08,343	00:00:01,890
	Nombre de noeuds	377304645	579555	14548915	4306182	9250397	15317014	579555	4305468	1366102
	Nombre de tests de contraintes	13100023708	12980037	289730424	116327221	188636086	305816752	12980037	116308473	27883498
grille2_5x5.fut	Temps	00:00:00,031	00:00:00,016	00:00:00,000	00:00:00,000	00:00:00,015	00:00:00,000	00:00:00,031	00:00:00,000	00:00:00,015
	Nombre de noeuds	78311	31578	731	731	20259	729	31578	729	31578
	Nombre de tests de contraintes	1413462	362790	10034	10034	359124	10007	362790	10007	362790
grille2_6x6.fut	Temps	00:00:00,062	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000	00:00:00,000
	Nombre de noeuds	118872	6196	313	90	6196	705	6196	145	6196
	Nombre de tests de contraintes	2901979	99512	4210	1235	99512	9530	99512	2099	99512
grille2_7x7.fut	Temps	00:11:17,281	00:00:23,421	00:00:10,703	00:00:00,281	00:01:27,500	00:00:09,218	00:00:33,000	00:00:00,218	00:01:32,703
	Nombre de noeuds	1113976918	25313978	9302442	153322	70908029	8999427	25313978	143924	87803810
	Nombre de tests de contraintes	2113400272	467047729	15039117	3514578	1167151008	144225068	467047729	3300218	1442991866
grille2_8x8.fut	Temps	00:00:07,515	00:00:00,250	00:00:00,281	00:00:01,281	00:01:19,265	00:00:00,171	00:00:01,156	00:00:01,156	00:00:35,063
	Nombre de noeuds	11505551	254279	192090	728753	54318763	134218	731347	731347	26973425
	Nombre de tests de contraintes	427562130	5419535	3919736	16936872	1038052964	2758073	16996239	16992623	514544990

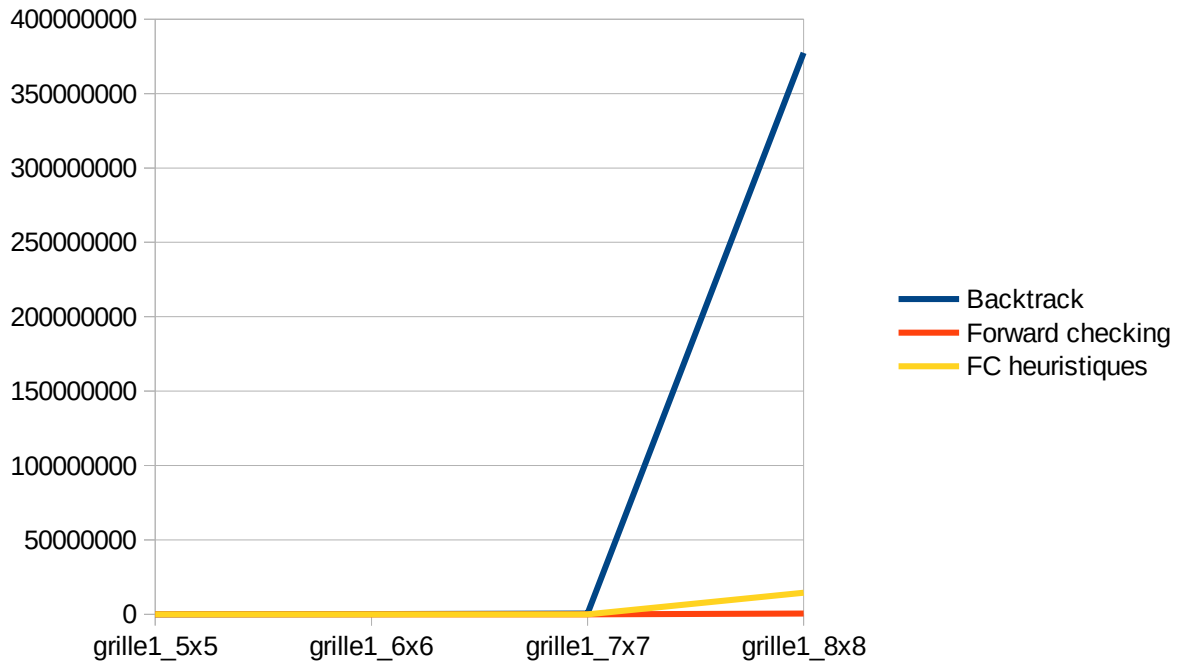
Comparaison entre les différents algorithmes

Grilles 1

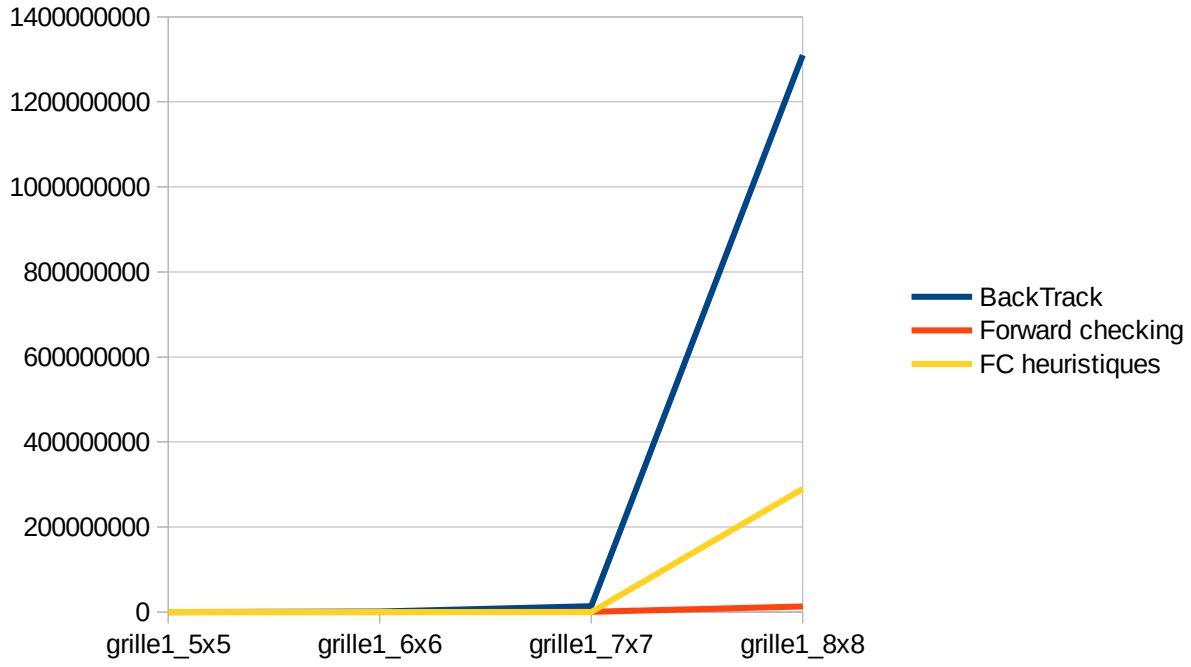
Temps



Nombre de nœuds dans l'arbre correspondant

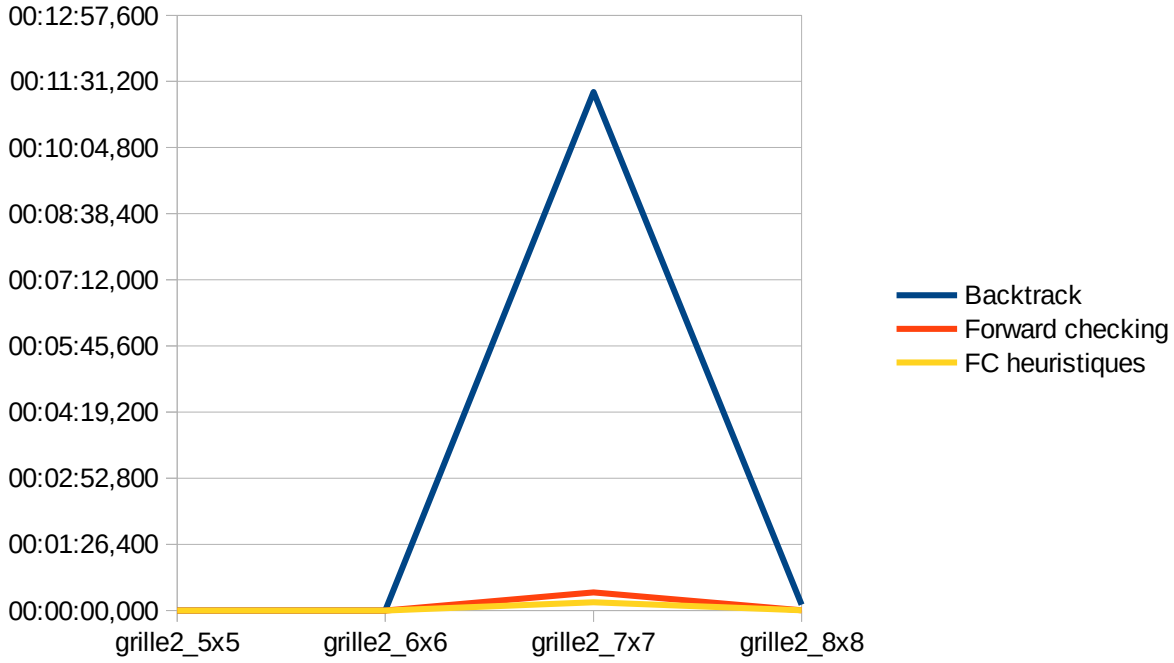


Nombre de test de contraintes

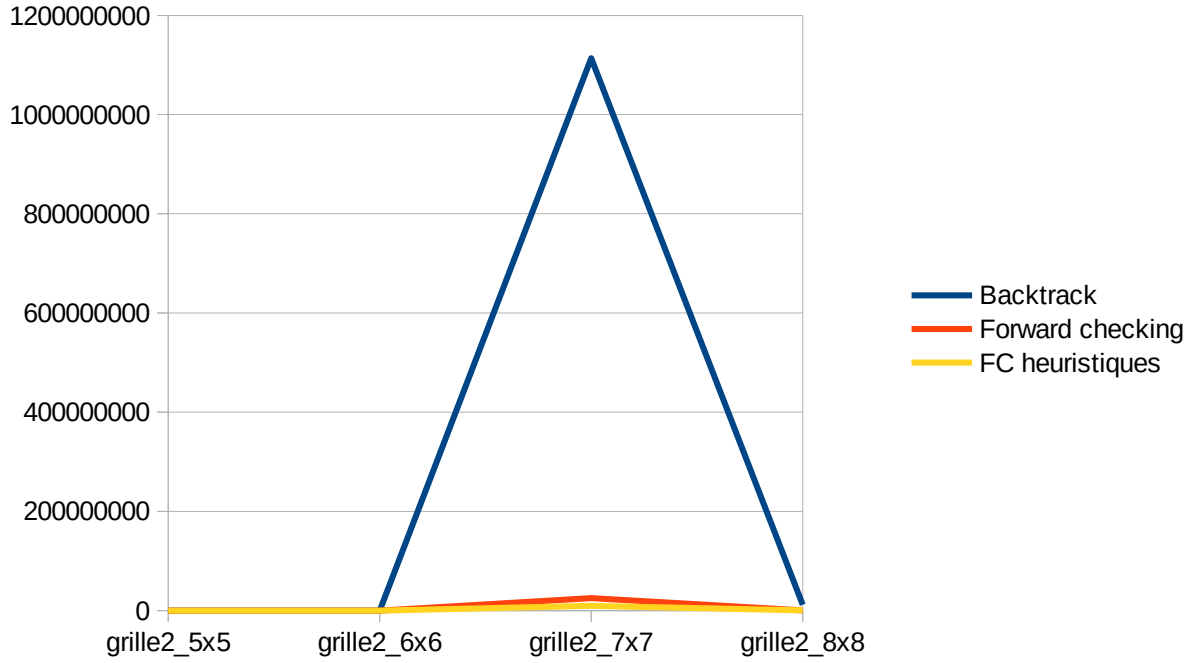


Grilles 2

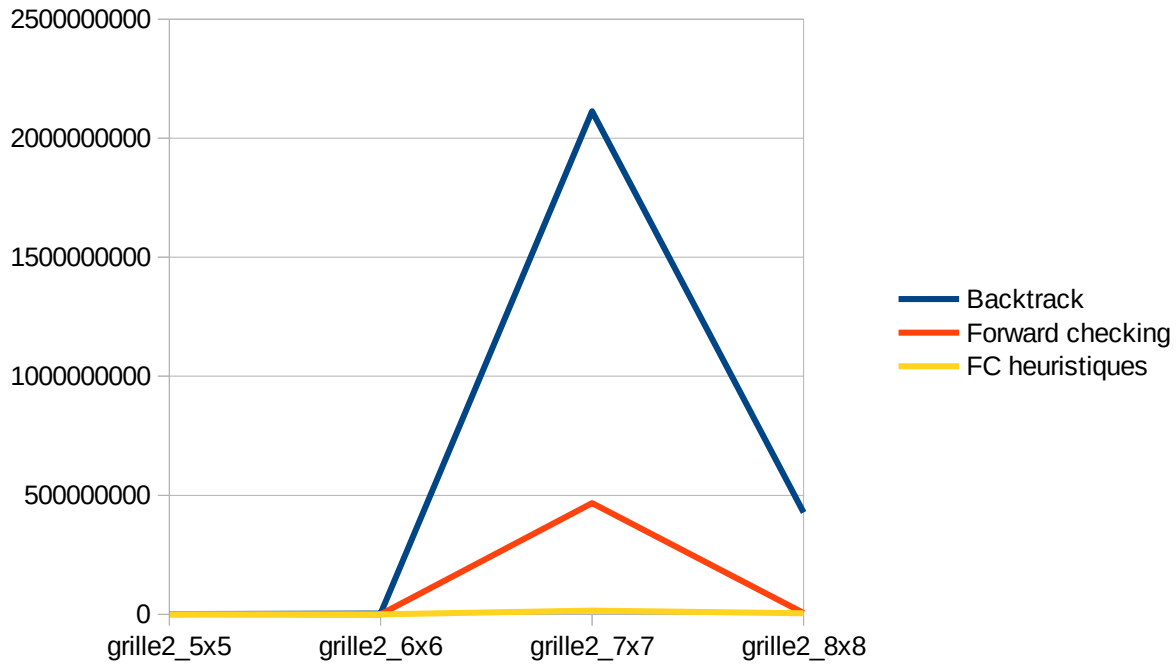
Temps



Nombre de nœuds dans l'arbre correspondant



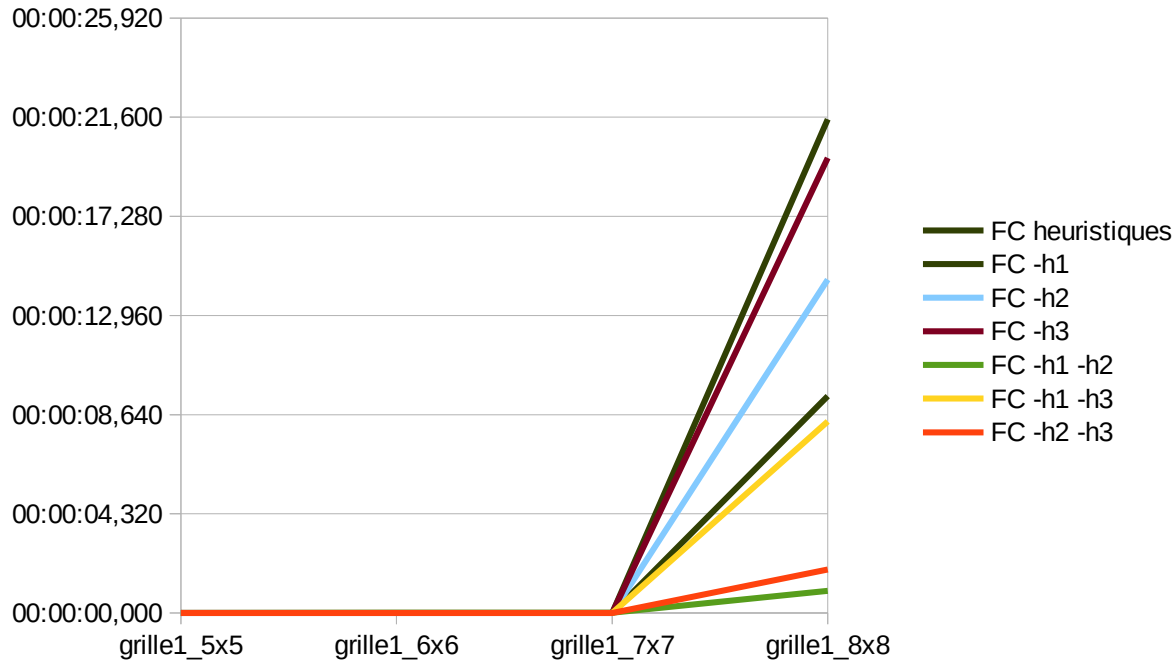
Nombre de test de contraintes



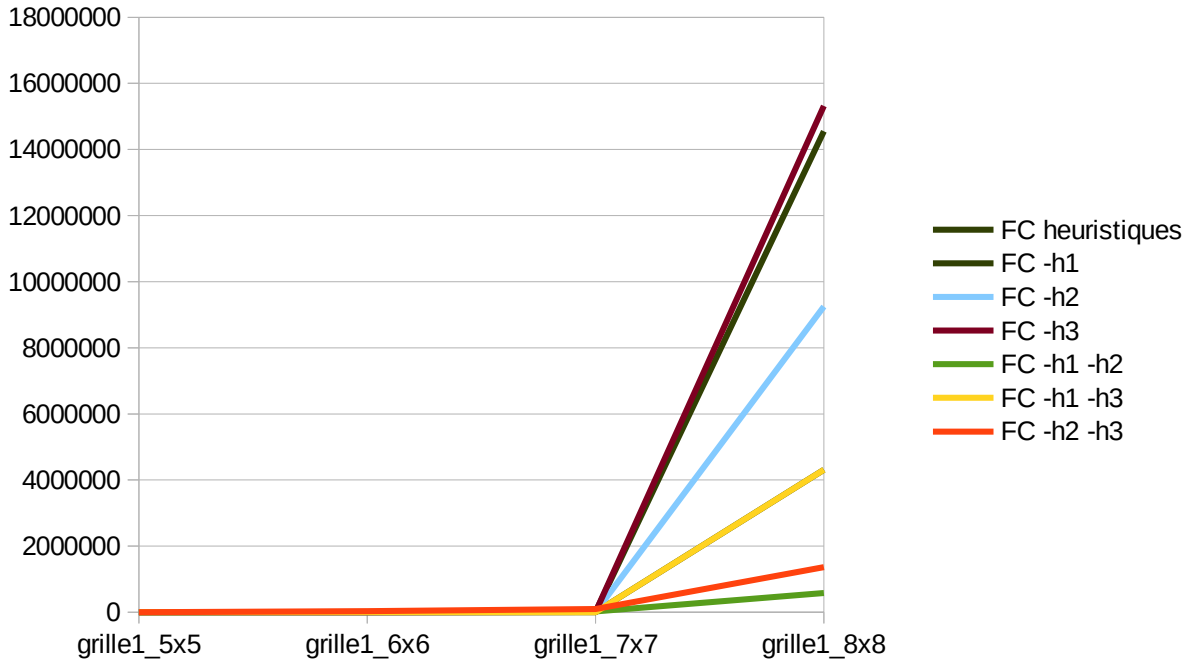
Comparaison entre les heuristiques

Grilles 1

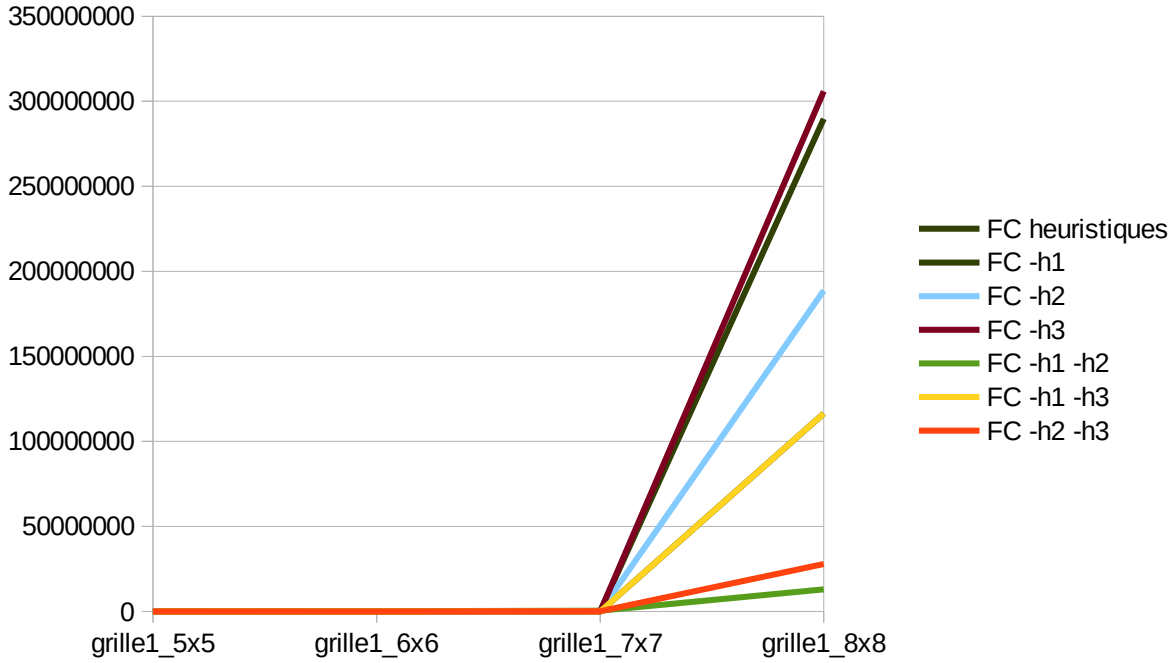
Temps



Nombre de nœuds dans l'arbre correspondant

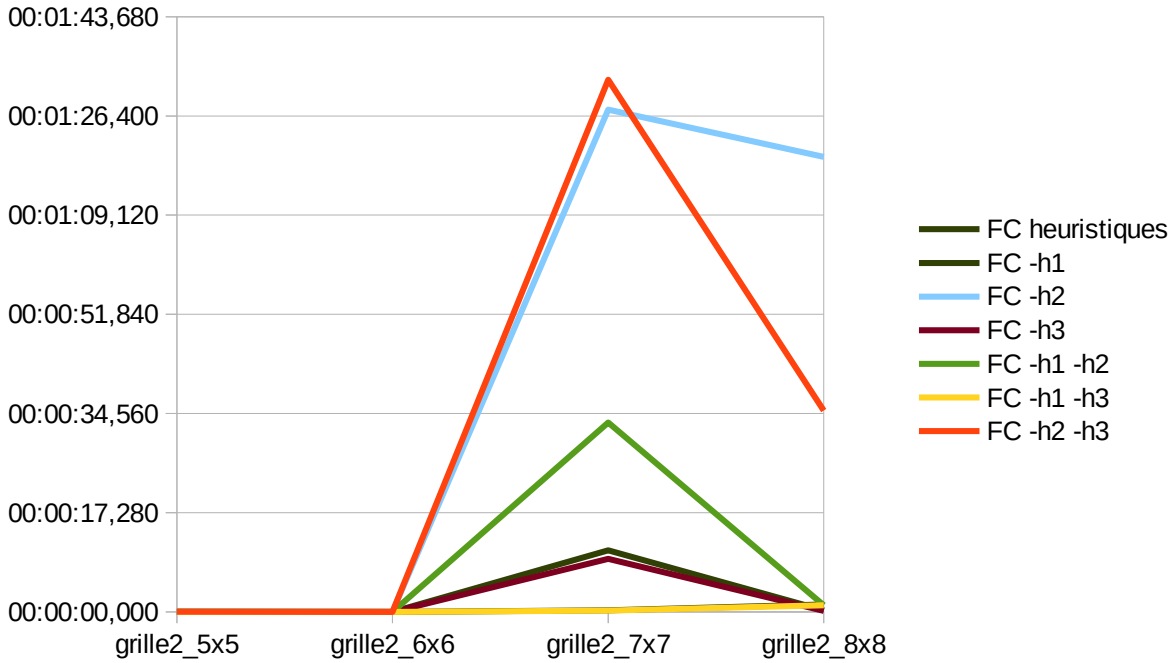


Nombre de tests de contraintes

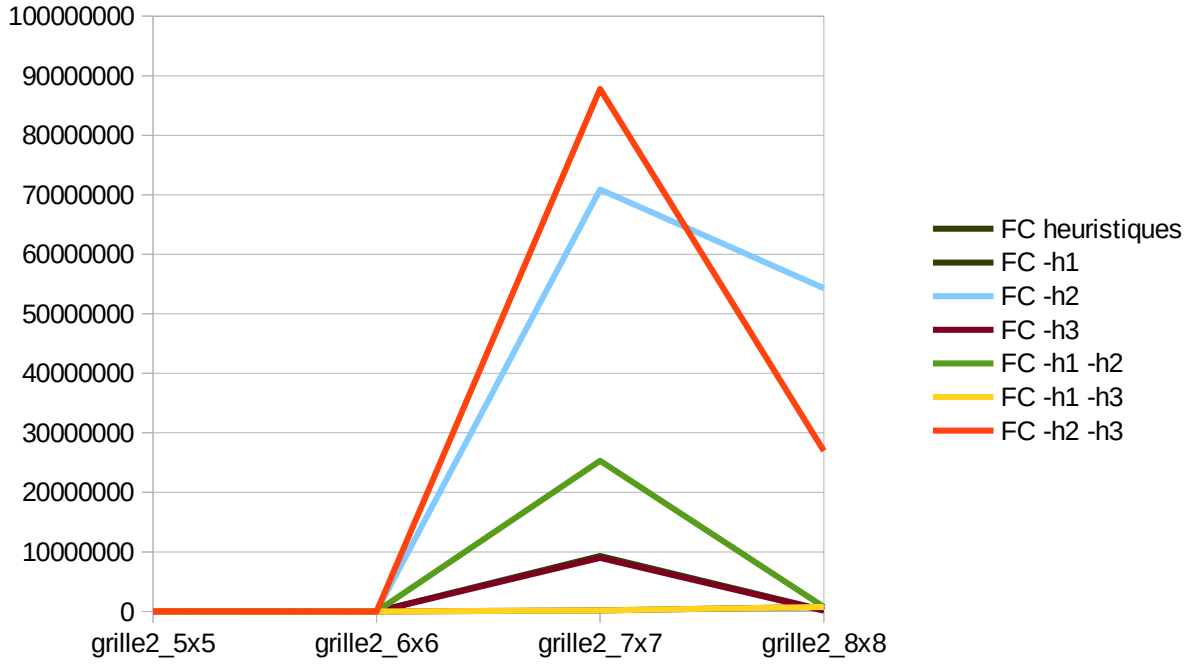


Grilles 2

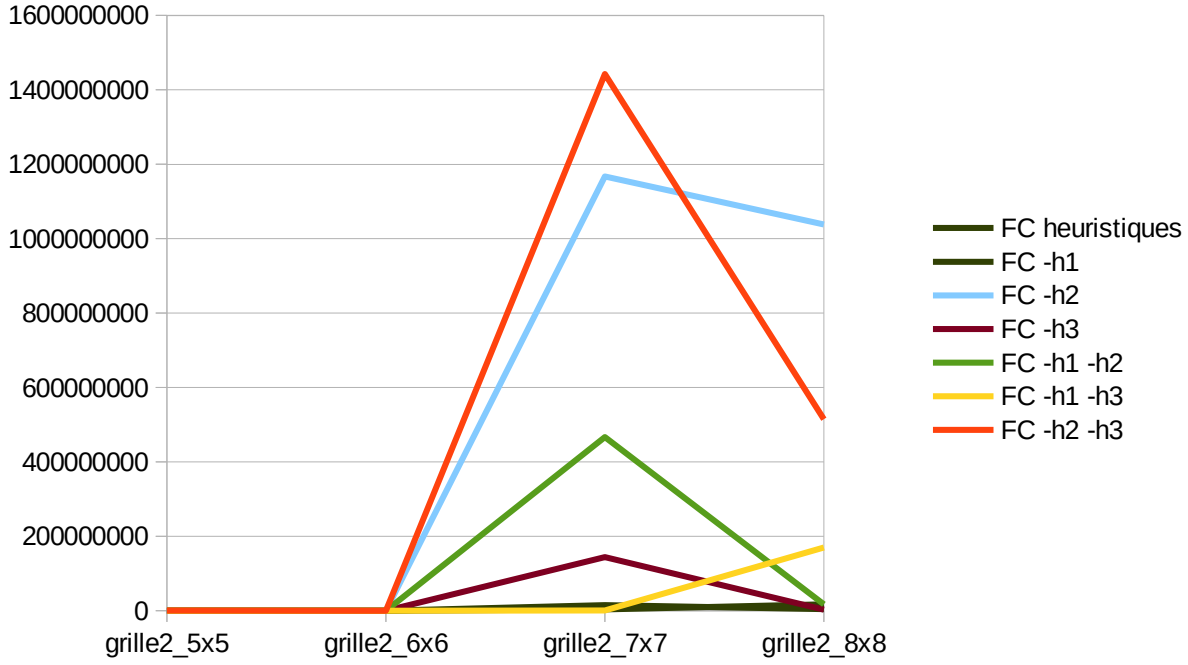
Temps



Nombre de nœuds dans l'arbre correspondant



Nombre de test de contrainte



Interprétations des résultats

Algorithmes

Comme on peut le voir, backtrack est clairement l'algorithme le moins efficace et de loin.

En revanche, entre forward checking et la version avec heuristiques, on voit que sur les grilles simples (grilles 1), forward checking est plus efficace tandis qu'avec les heuristiques, c'est sur les grilles complexes (grilles 2) que c'est le plus efficace.

Cela s'explique par le fait que les heuristiques sont prévus pour aider à résoudre les grilles avec beaucoup de contraintes et notamment avec les contraintes de nombres (nombres déjà placés).

Les heuristiques sont donc réellement efficaces surtout sur le nombre de test de contraintes.

Heuristiques

Il faut rappeler que les options (tel « -h1 ») **désactive** l'heuristique. Ainsi si les valeurs sont plus élevées que celles avec toutes les heuristiques activées, alors c'est que l'heuristique fut efficace.

Ainsi, on peut voir que pour les grilles « simples », les heuristiques de choix de variables (variables avec plus petit domaine et variable avec contraintes autres que DIF) font perdre en efficacité. Cela s'explique par le fait que les grilles « simples » possèdent peu de contraintes d'inférioriété ou de supériorité et pas de contraintes de nombre déjà placé.

En revanche, l'heuristique de choix de valeur augmente grandement l'efficacité de forward checking (-h1 -h2). En effet, sans les autres contraintes, la solution est simplement de toujours prendre une valeur différentes de celles sur la ligne et la colonne, donc prendre la valeur la moins utilisée semble être la bonne solution. Seule, cette heuristique est donc très efficace dans ce cas.

Par contre, pour les grilles « complexes », cela est différent. En effet, désactiver l'heuristique de choix de variable sur la taille du domaine ainsi que de choix de valeur (-h1 -h3) ou séparément (-h1 ou -h3) semble améliorer l'efficacité de forward checking. L'heuristique de choix de variable par contraintes autres que différence semble donc grandement en améliorer l'efficacité.

Cela s'explique par le fait que s'il y a plus de contraintes de supériorité ou d'inférioriété, alors en prenant ces variables dès le début, on peut changer beaucoup plus de valeurs dans les domaines des autres variables et ainsi arriver à un échec plus vite.

Pour l'heuristique de choix de valeur, son inefficacité s'explique de la même façon que précédemment : avec plus de contraintes de supériorité ou d'inférioriété, ainsi que des contraintes de nombre déjà placées, choisir simplement les valeurs les moins utilisées n'amène pas vers la solution de la grilles.

Enfin, pour l’heuristique de choix de variable selon la taille du domaine, actuellement, seules les variables avec une contraintes de nombre déjà placé entre dans cette heuristique (et ce à cause du problème de pointeurs lors du tri du tableau). Ainsi, il semble que sur ces grilles, cela ne soit pas rentable.

Conclusion

Pour conclure, on sait désormais que pour les grilles « simples », les heuristiques de choix de variable ne sont pas rentables et forward checking avec l’heuristique de choix de valeur est le plus efficace.

Pour les heuristiques, l’heuristique de choix de variable selon les contraintes autres que de supériorité ou d’infériorité est la plus efficace et dans le plus de cas.

Pour le choix de valeur, cela n’est rentable que pour les grilles « simples ».

Pour le choix de variable selon la taille du domaine, actuellement, cela n’est pas très rentable. En revanche, nous avons pu expérimenter en triant ce tableau au fur et à mesure à l’aide du quicksort. Les résultats sont que sur les grilles « complexes » avec une solution valide, cette heuristique est très efficace. Mais comme nous avons ce problème de pointeur lors du tri au fur et à mesure, on ne peut l’exploiter pleinement car la solution est fausse sur certaines grilles.