Une stratégie d'anticipation pour la planification par recherche heuristique

A lookahead strategy for heuristic search planning

Vincent Vidal

CRIL - Université d'Artois
rue de l'Université - SP 16
62307 Lens Cedex, France
vidal@cril.univ-artois.fr

Résumé

Les plans relaxés sont utilisés dans le planificateur par recherche heuristique FF pour calculer une heuristique numérique et extraire les actions "utiles". Nous présentons une nouvelle manière d'extraire des informations d'un plan relaxé et d'employer les actions utiles, en considérant la grande qualité de ces plans relaxés dans de nombreux domaines. Pour chaque état évalué, nous employons les actions du plan relaxé retourné par la fonction de calcul de l'heuristique afin de trouver le début d'un plan valide conduisant à un état atteignable. Nous utilisons cette stratégie d'anticipation dans un algorithme de recherche de type meilleur-d'abord, modifié afin de prendre en compte les actions utiles. Dans de nombreux domaines de planification, les performances de la planification par recherche heuristique et la taille des problèmes pouvant être résolus sont très fortement améliorés, tandis que dans d'autres domaines plus "difficiles" ces stratégies restent intéressantes même si elles dégradent parfois un peu la qualité en nombre d'actions des plans produits.

Mots Clef

Planification, recherche heuristique, graphe de planification, FF.

Abstract

Relaxed plans are used in the heuristic search planner FF for computing a numerical heuristic and extracting helpful actions. We present a novel way for extracting information from the relaxed plan and for dealing with helpful actions, by considering the high quality of the relaxed plans in numerous domains. For each evaluated state, we employ actions from these plans in order to find the beginning of a valid plan that can lead to a reachable state. We use this lookahead strategy in a complete best-first search algorithm, modified in order to take into account helpful actions. In numerous planning domains, the performance of heuristic search planning and the size of the problems that can be handled have been drastically improved, while in more "difficult" domains these strategies remain interesting even if they sometimes degrade plan quality.

Keywords

Planning, heuristic search, planning graph, FF.

1 Introduction

La planification par recherche heuristique a prouvé qu'elle était un cadre de recherche intéressant pour la planification classique non-optimale de type STRIPS [6], depuis l'arrivée de planificateurs capables de dépasser, dans la plupart des domaines classiques, les performances des planificateurs précédents Graphplan [1, 2], Blackbox [9], IPP [10], STAN [11], SGP [14], ... Bien que la plupart des ces derniers calculent des plans parallèles optimaux en nombre de niveaux d'actions parallèles, ils n'offrent pas plus de garantie en terme d'optimalité en nombre d'actions que les planificateurs fonctionnant par recherche heuristique les plus performants. C'est une des raisons pour lesquelles l'attention de la communauté scientifique de planification s'est récemment tourné vers ce cadre de recherche, plus prometteur en terme de performances pour la planification non optimale.

La planification par recherche heuristique, initiée – voire même redécouverte, cf. [13] – par les travaux sur les planificateurs ASP [5], HSP et HSPr [3, 4], a conduit à certains des planificateurs les plus performants, comme cela a été démontré lors des deux dernières éditions de la compétition internationale de planification par les planificateurs HSP2 [4], FF [8] et AltAlt [12]. FF a été en particulier récompensé pour ses performances remarquables lors de la 2$^{\text{ème}}$ compétition internationale de planification[1], et a été également le plus performant lors de la 3$^{\text{ème}}$ compétition internationale[2] pour les catégories dans lesquelles il concourrait.

[1] Site internet de la 2$^{\text{ème}}$ IPC (International Planning Competition) : http ://www.cs.toronto.edu/aips2000/

[2] Site internet de la 3$^{\text{ème}}$ IPC : http ://www.dur.ac.uk/d.p.long/competition.html

Nous nous intéressons dans cet article à une technique de calcul d'heuristique introduite dans le planificateur FF, basée sur l'extraction d'une solution d'un graphe de planification calculé pour le problème relaxé obtenu en ignorant les retraits des actions. Ce calcul est exécuté en temps et en espace polynomial par rapport aux données du problème (nombre de fluents, nombre d'actions…). Le nombre d'actions du plan relaxé pour un état constitue pour FF la valeur numérique de l'heuristique pour cet état, et représente une estimation de la longueur du plan nécessaire pour atteindre les buts. Cette heuristique est utilisée dans un algorithme de recherche en chaînage avant pour évaluer chacun des états rencontrés. Une autre information est dérivée du graphe de planification relaxé et de sa solution : l'ensemble des actions dites "*utiles*". Ce sont les actions du graphe relaxé exécutables dans l'état pour lequel l'heuristique est calculée, augmentées dans FF par toutes les actions applicables dans cet état et produisant des fluents qui ont été déterminés comme étant des sous-buts au premier niveau du graphe de planification lors de l'extraction du plan. Ces actions permettent à FF de concentrer ses efforts sur des branches du graphe de recherche plus prometteuses que les autres, en oubliant les actions qui ne sont pas "utiles" par une variation de l'algorithme de recherche locale "hill-climbing". Quand ce dernier ne parvient pas à trouver une solution, la recherche est relancée depuis le début par un algorithme de recherche complet de type meilleur-d'abord. Le bénéfice pouvant être apporté par les actions utiles et la recherche locale est alors perdu.

Nous introduisons un nouvelle méthode pour extraire des informations intéressantes du calcul de l'heuristique et pour utiliser les actions utiles, en considérant la grande qualité des plans relaxés obtenus par la fonction heuristique dans de nombreux domaines. En effet, le début de ces plans peut souvent être étendu en des plans solutions du problème initial (non relaxé), et un grand nombre d'actions présentes dans ces plans se retrouvent effectivement dans les plans solutions. Nous présentons dans cet article un algorithme permettant de combiner un certain nombre d'actions d'un plan relaxé, afin de trouver le début d'un plan valide permettant de calculer un état atteignable. Grâce à la bonne qualité des plans relaxés, ces états se rapprochent fréquemment d'un état solution contenant les buts du problème. Les *états anticipés* ainsi calculés sont alors ajoutés à la liste des noeuds pouvant être développés, dont le choix est toujours laissé à la valeur numérique de l'heuristique. La meilleure stratégie que nous ayons trouvée pour calculer ces états supplémentaires est d'utiliser autant que possible d'actions du plan relaxé, et de calculer ces états anticipés pour chaque nouveau noeud ajouté à la liste des noeuds développés, même lorsqu'un de ce nouveaux noeuds est lui-même construit à partir d'un état anticipé.

Cette stratégie d'anticipation peut être utilisée dans différents algorithmes de recherche. Nous proposons une modification d'un algorithme de recherche classique de type meilleur-d'abord d'une façon qui préserve sa complétude. En effet, rajouter des noeuds formés à partir de ces états anticipés dans la liste des noeuds devant être développés ne modifie ni la complétude, ni la terminaison de l'algorithme, si ces états sont bien atteignables et que l'on ne développe pas un état qui a déjà été développé. Le facteur de branchement est légèrement augmenté par cet ajout, mais en pratique les performances sont souvent fortement améliorées.

En plus de cette stratégie d'anticipation, nous proposons une nouvelle utilisation possible des actions "utiles" qui préserve la complétude contrairement à l'utilisation qui en est faite dans le planificateur FF. Dans ce dernier, les actions qui ne sont pas considérées comme utiles sont perdues : cela rend l'algorithme de recherche incomplet. Pour éviter cela, nous modifions plusieurs aspects de cet algorithme. Quand un état E est évalué, deux nouveaux noeuds sont ajoutés dans la liste des noeuds devant être développés (appelée par la suite la liste $open$) : un noeud qui contient les actions utiles, qui sont les actions appartenant au plan relaxé exécutables dans E, et un noeud qui contient les actions applicables dans E n'appartenant pas au plan relaxé (appelées par la suite *actions de secours*). Le type des actions attachées au noeud (utiles ou secours) est conservé par ce noeud. Nous ajoutons alors un critère au mécanisme de sélection d'un noeud dans la liste $open$, qui donne la préférence à un noeud contenant des actions utiles par rapport à un noeud contenant des actions de secours, quelle que soit la valeur numérique de l'heuristique pour ces états. Comme aucune action n'est perdue comme dans FF et qu'aucun état n'est perdu, la complétude de l'algorithme de recherche est préservée.

Notre évaluation expérimentale de l'utilisation de cette stratégie d'anticipation par un algorithme complet de type meilleur-d'abord bénéficiant des actions utiles, conduite sur une large palette de domaines de tests, démontre que les performances en terme de temps de calcul et de taille des problème pouvant être résolus ont été très fortement améliorées. La prise en compte des action utiles rend un algorithme de recherche complet toujours plus rapide, tandis que la stratégie d'anticipation le rend capable de résoudre des problèmes de taille extrêmement importante (par rapport à l'état de l'art actuel en planification) dans de nombreux domaines. Un inconvénient de cette stratégie d'anticipation est de parfois dégrader la qualité du plan solution en nombre d'actions. Mais le compromis que l'on doit souvent faire entre temps de calcul et qualité de la solution, même pour des problèmes "difficiles" dont la longueur du plan est substantiellement augmenté, semble toujours pencher en faveur de notre stratégie d'anticipation.

L'article est organisé de la façon suivante. Après avoir donné quelques définitions classiques en Section 2, nous présentons les idées principales en Section 3 : le calcul et l'utilisation des états anticipés, et l'utilisation des actions utiles par un algorithme de recherche complet. Nous décrivons ensuite les algorithmes utilisant ces concepts en Section 4. Nous présentons finalement une évaluation expérimentale de notre travail en Section 5, avant de conclure en Section 6.

2 Définitions

Un *état* est un ensemble fini de formules atomiques de base (i.e. sans symbole de variable) aussi appelées *fluents*. Les *actions* sont des actions classiques de type STRIPS. Soit a une action ; $Prec(a)$, $Add(a)$ et $Del(a)$ sont des ensembles de fluents et dénotent respectivement les préconditions, ajouts et retraits de l'action a. Un *problème de planification* est un triplet $\langle A, I, B \rangle$ où A est un ensemble d'actions, I est un ensemble de fluents dénotant l'état initial et B est un ensemble de fluents dénotant le but. L'*application* d'une action a sur un état E (noté $E \uparrow a$) est possible ssi $Prec(a) \subseteq E$, et l'état résultant est défini par $E \uparrow a = (E \backslash Del(A)) \cup Add(a)$. Un *plan* est une séquence d'actions. Soit un plan $P = \langle a_1, a_2, \ldots, a_n \rangle$; P est *valide* pour un état E ssi a_1 est applicable à E et conduit à un état E_1, a_2 est applicable à E_1 et conduit à un état E_2, ..., a_n est applicable à E_{n-1} et conduit à un état E_n. Dans ce cas, E_n est dit *atteignable* depuis E pour P et P est un *plan solution* ssi $B \subseteq E_n$. $Premier(P)$, $Reste(P)$ et $Long(P)$ dénotent respectivement la première action de P (ici a_1), P privé de la première action (ici la séquence $\langle a_2, \ldots, a_n \rangle$), et la longueur de P en nombre d'actions (ici n).

3 Les idées principales

3.1 États et plans anticipés

Pour les algorithmes classiques de recherche dans les espaces d'états, un noeud dans le graphe de recherche représente un état et un arc depuis ce noeud représente l'application d'une action à cet état, conduisant à un nouvel état. Afin d'assurer la complétude, toutes les actions applicables à cet état doivent être considérées. L'ordre dans lequel ces actions sont alors considérées pour développer les états dépend de la stratégie choisie : profondeur d'abord, largeur d'abord, meilleur-d'abord (suivant la valeur d'une heuristique), ...

Le problème essentiel pour la planification par recherche heuristique est alors bien sûr la sélection du meilleur noeud possible à développer. Les difficultés commencent quand les valeurs heuristiques des meilleurs noeuds sont proches les unes des autres : aucun état ne se distingue particulièrement des autres.

Imaginons maintenant que pour chaque état évalué E, nous connaissions un plan valide P pouvant être appliqué à E et conduisant à un état E' plus proche d'un état but que les descendants directs de E (après l'application d'une seule action). Il serait alors intéressant d'appliquer P à E, et d'utiliser l'état résultant E' comme un nouveau noeud dans la recherche. Cet état serait alors considéré comme un nouveau descendant de E.

Nous avons alors deux types d'arcs dans le graphe de recherche : ceux qui viennent de l'application directe d'une action à un état, et ceux qui viennent de l'application d'un plan valide à un état E et conduisent à un état E' atteignable depuis E. Nous appellerons de tels états des états *anticipés*, parce qu'il sont calculés par l'application d'un plan à l'état E attaché à un noeud n_E mais sont considérés dans le graphe de recherche comme des descendants directs de n_E. Les noeuds créés pour des états anticipés seront appelés des *noeuds anticipés*. Les plans attachés aux arcs conduisant à des noeuds anticipés seront appelés des *plans anticipés*. Une fois qu'un état but est trouvé, le plan solution est alors la concaténation d'actions uniques pour les arcs conduisant à des noeuds classiques et de plans anticipés pour des arcs conduisant à des noeuds anticipés.

La détermination d'une valeur heuristique pour chaque état, comme elle est faite dans le planificateur FF, offre un moyen de calculer de tels plans anticipés. FF crée un graphe de planification relaxé pour chaque état rencontré E, en utilisant le problème relaxé obtenu en ignorant les retraits des actions et en utilisant E comme état initial. Un plan relaxé est alors extrait en temps et en espace polynomial de ce graphe de planification. La longueur du plan en nombre d'actions correspond à la valeur heuristique de l'état pour lequel il est calculé (il faut noter que cette longueur surestime en général la longueur réelle du plan nécessaire, rendant cette heuristique non admissible). En général, le plan relaxé pour un état E n'est pas valide pour E, car les retraits des actions sont ignorés durant son calcul : les interactions négatives entre actions ne sont pas considérées, et ainsi une action peut retirer un but ou un fluent requis en précondition par des actions qui suivent dans le plan relaxé. Cependant, les actions d'un plan relaxé sont utilisées parce qu'elle produisent des fluents qui peuvent être intéressants pour produire les buts, donc des actions de ces plans peuvent être intéressantes pour produire le plan solution d'un problème. Dans de nombreux domaines, nous pouvons observer que les plans relaxés ont une bonne qualité car ils contiennent beaucoup d'actions qui appartiennent au plan solution. Nous proposons en Section 4 un algorithme pour calculer des plans anticipés à partir de ces plans relaxés.

La complétude d'un algorithme de recherche sera préservée par l'utilisation de cette technique d'anticipation, puisque l'on ne fait que développer des noeuds supplémentaires qui auraient dû être atteints plus tard dans le graphe de recherche. On ne fait que modifier la façon dont l'espace de recherche est exploré. De même, la correction de l'algorithme sera préservée si les plans anticipés que l'on calcule mènent bien à des états anticipés atteignables depuis l'état initial. La terminaison n'est pas affectée non plus, si on ne développe pas plusieurs fois des noeuds déjà rencontrés.

3.2 Utilisation des actions utiles : la stratégie "optimiste"

Dans les algorithmes de recherche classiques, toutes les actions qui peuvent être appliquées à un noeud sont considérées de la même façon : les états auxquels ils conduisent sont évalués par une fonction de calcul d'heuristique et sont ordonnés suivant la valeur calculée, mais il n'y a pas de notion de préférence entre les actions elles-mêmes. Une telle notion de préférence durant la recherche a été introduite dans le planificateur FF [8], avec le concept d'actions utiles. Une fois que le graphe de planification relaxé pour un état E est calculé et qu'un plan relaxé en est extrait, les actions applicables à E sont considérées comme étant utiles tandis que les autres actions sont oubliées par l'algorithme de recherche locale. Mais cette stratégie s'avérant trop restrictive, l'ensemble des actions utiles dans FF est augmenté par toutes les actions applicables à E qui produisent des fluents qui ont été considérés comme des sous-buts au premier niveau du graphe pendant l'extraction du plan relaxé. L'inconvénient principal de cette stratégie, telle qu'elle est utilisée dans FF, est qu'elle ne préserve pas la complétude : les actions applicables à E qui ne sont pas considérées comme utiles sont perdues. En cas d'échec, la recherche est alors reprise depuis le début par un algorithme de type meilleur-d'abord complet, ne tirant pas partie des actions utiles.

Nous présentons dans cet article un moyen d'utiliser la notion d'actions utiles par des algorithmes de recherche complets, que nous qualifierons d'*optimistes* car ils accordent une confiance maximale aux informations fournies par le calcul de l'heuristique. Le principe en est le suivant :

- Plusieurs classes d'actions sont créées. Par la suite, nous n'utiliserons que deux d'entre elles : les actions utiles (les plus restrictives : celles qui appartiennent au plan relaxé et sont applicables dans l'état pour lequel on calcule l'heuristique), et les actions de secours (toutes les actions qui ne sont pas utiles).
- Quand un nouvel état E est évalué, la fonction de calcul de l'heuristique retourne une estimation numérique de l'état ainsi que les actions applicables à cet état partitionnées dans leurs différentes classes. Il faut noter qu'une même action peut être utile pour un état donné, mais être de secours pour un autre état. Pour chaque classe, un noeud est créé pour l'état E, auquel on attache les actions de la classe considérée retournées par la fonction de calcul de l'heuristique.
- Lorsqu'on sélectionne un noeud à développer, la classe des actions qui lui sont attachées est prise en compte : dans notre algorithme *optimiste*, c'est le critère le plus important. Les noeuds contenant des actions utiles seront toujours préférés aux noeuds contenant des actions de secours, quelle que soit la valeur de l'heuristique.

Aucune information n'est perdue par ce procédé. La façon dont les noeuds sont développés est simplement modifiée : un état E est développé d'abord avec les actions utiles, d'autres noeuds peuvent ensuite être développés, et E peut éventuellement être développé avec les actions de secours. Comme l'union des actions utiles et des actions de secours pour un noeud est égal à l'ensemble des actions applicables à ce noeud, la complétude d'un algorithme de recherche et la correction des plans trouvés sont conservées.

4 Description des algorithmes

Dans cette section, nous décrivons les algorithmes de notre planificateur et discutons les principales différences avec le planificateur FF, sur lequel est basé notre travail. Les fonctions principales de l'algorithme sont les suivantes :

planifier$()$: C'est la fonction principale (voir Figure 1). D'abord, la fonction **calculer_noeud** est appelée sur l'état initial du problème et le plan vide : l'état initial est évalué par la fonction heuristique, et un noeud est créé et ajouté à la liste $open$. Les noeuds de cette liste ont la structure suivante : $\langle E, P, A, h, t\rangle$, avec :
- E est un état,
- P est le plan qui a conduit à E depuis l'état initial,
- A est l'ensemble des actions applicables à E,
- h est la valeur heuristique de E (le nombre d'actions du plan relaxé calculé pour E),
- t est le type des actions de A : $utiles$ ou $secours$.

Commence ensuite une boucle qui choisit le meilleur noeud et le retire de la liste $open$ (fonction **pop_meilleur_noeud**), et le développe jusqu'à ce qu'un plan soit trouvé ou que la liste $open$ soit vide. En contraste avec les algorithmes de recherche standards, les actions choisies pour être appliquées sur l'état attaché à ce noeud sont déjà connues, car elles font parties des informations du noeud. Elles viennent du calcul de l'heuristique par la fonction **calculer_heuristique** appelée par **calculer_noeud**, qui retourne les ensembles d'actions utiles et d'actions de secours.

pop_meilleur_noeud$()$: retourne le meilleur noeud de la liste $open$. Les noeuds sont comparés suivant trois critères d'importance décroissante. Soit $N = \langle E, P, A, h, t\rangle$ un noeud de la liste $open$. Le premier critère est le type t des actions de A : $utiles$ est préféré à $secours$. En cas d'égalité de ces types, le critère suivant minimise la fonction heuristique $f(E) = W \times h + Long(P)$, comme dans les algorithmes de recherche de type *WA**. Dans notre implémentation actuelle, nous utilisons $W = 3$ (choix motivé par les expérimentations). En cas d'égalité, le troisième critère minimise la longueur du plan P qui conduit à E.

calculer_noeud(E, P) : elle est appelée par **planifier** sur l'état initial et le plan vide, ou par **planifier** ou elle-même sur un état nouvellement créé E et le plan P qui a conduit à cet état. Les appels par **planifier** viennent de l'état initial ou de la sélection d'un noeud dans la liste $open$ et de l'application d'une action sur un état donné. Les appels par elle-même viennent du calcul d'un plan anticipé et de son application sur un état donné.

Si l'état E appartient à la liste $close$ (les noeuds déjà développés) ou est un état but, la fonction se termine. Sinon, E est évalué par la fonction **calculer_heuristique**, qui retourne un plan relaxé, un ensemble d'actions utiles et un ensemble d'actions de secours.

Cette opération est effectuée la première fois avec un sous-ensembles des actions du problème, que nous appellerons

actions *but-préférées*. Ce sont les actions qui ne retirent pas de fluent qui à la fois appartient au but, et n'appartient pas à l'état initial. Ceci peut se justifier intuitivement par le fait qu'une fois qu'un but est produit par une action, on ne souhaite pas utiliser une action qui le retire. Ceci étant un peu trop restrictif, on autorise quand même l'utilisation d'actions qui retirent un but qui appartient aussi à l'état initial. Bien que n'ayant pas de véritable justification théorique, la distinction entre les actions but-préférées et les autres actions apporte beaucoup d'un point de vue pratique.
Si un plan relaxé est trouvé, deux noeuds sont ajoutés à la liste $open$: un pour les actions utiles et un pour les actions de secours. Un plan anticipé est alors recherché par la fonction **calcul_anticipation**, et **calculer_noeud** est appelée à nouveau sur l'état résultant de l'application de ce plan anticipé sur E (si ce plan anticipé contient au moins deux actions).
Si aucun plan anticipé ne peut être trouvé avec les actions but-préférées, l'heuristique est évaluée à nouveau avec l'ensemble complet des actions. Dans ce cas, on considère que E a un intérêt plus faible : si un plan relaxé est trouvé, seulement un noeud est ajouté à la liste $open$: les actions utiles sont considérées comme des actions de secours, et on ne calcule pas de plan anticipé.

calculer_heuristique(E, A) : cette fonction calcule la valeur heuristique d'un état E d'une manière similaire FF. Un graphe relaxé est d'abord créé, en utilisant les actions de A. Ce paramètre permet de réduire les actions à utiliser aux actions but-préférées : ceci se montre utile dans plusieurs domaines. Une fois créé, un plan relaxé est extrait du graphe.
Dans notre implémentation actuelle, il y a deux différences essentielles par rapport à FF. La première tient à la façon dont les actions sont utilisées dans le plan relaxé. Dans FF, quand une action est utilisée au niveau i, ses ajouts sont marqués comme étant vrais au niveau i (comme dans Graphplan), mais aussi au niveau $i - 1$. Par conséquent, une précondition d'une action requise par une autre action du même niveau ne considérera pas ces fluents comme des nouveaux sous-buts. Dans notre implémentation, les ajouts d'une action sont marqués vrais au niveau i, mais ses préconditions sont requises au niveau i et non au premier niveau dans lequel cette action apparaît, comme dans FF. Une précondition d'une action à un niveau donné peut ainsi être produite par une action du même niveau, ce qui fait que le choix d'actions pour produire cette précondition est plus large.
La deuxième différence tient dans la façon dont les actions sont ajoutées au plan relaxé. Dans FF, les actions sont placées dans l'ordre dans lequel elles sont sélectionnées. Nous avons trouvé plus efficace en pratique d'utiliser l'algorithme suivant. Soit a une action, et $\langle a_1, a_2, \ldots, a_n\rangle$ un plan. a est ordonnée après a_1 ssi le niveau du sous-but pour lequel a a été sélectionnée est supérieur ou égal au niveau du sous-but pour lequel a_1 a été sélectionnée, et soit a retire une précondition de a_1 soit a_1 ne retire pas de précondition de a. Dans ce cas, le processus continue entre a et a_2, et ainsi de suite jusqu'à la fin du plan. Sinon, a est placée avant a_1.

soit $\Pi = \langle A, I, B\rangle$; /* problème de planification */
soit $ABP = \{a \in A \mid \forall f \in Del(a), f \notin (B \setminus I)\}$;
/* actions but-préférées */
soit $open = \varnothing$; /* liste $open$: noeuds à développer */
soit $close = \varnothing$; /* liste $close$: noeuds déjà développés */

fonction planifier ()
calculer_noeud$(I, \langle\rangle)$;
tantque $open \neq \varnothing$ **faire**
soit $\langle E, P, actions, h, type\_actions\rangle$ = pop_meilleur_noeud()
pour $a \in actions$ **faire**
calculer_noeud$(E \uparrow a, P \oplus \langle a\rangle)$
finpour
fintantque
fin

fonction calculer_noeud (E, P) /* E : état, P : plan, */
si $E \notin close$ **alors**
si $B \subseteq E$ **alors** écrire_plan_et_finir(P) **finsi** ;
$close \leftarrow close \cup \{E\}$;
soit $\langle PR, U, S\rangle$ = calculer_heuristique(E, ABP) ;
si $PR \neq fail$ **alors**
$open \leftarrow open \cup \{\langle E, P, U, Long(PR), utiles\rangle,$
$\langle E, P, S, Long(PR), secours\rangle\}$;
soit $\langle E', P'\rangle$ = calcul_anticipation(E, PR) ;
si $Long(P') \geq 2$ **alors**
calculer_noeud$(E', P \oplus P')$
finsi
sinon
soit $\langle PR, U, S\rangle$ = calculer_heuristique(E, A) ;
si $PR \neq fail$ **alors**
$open \leftarrow open \cup \{\langle E, P, Long(PR), U \cup S, secours\rangle\}$
finsi
finsi
finsi
fin

FIG. 1 – Algorithme de recherche meilleur-d'abord optimiste avec anticipation

Les différences entre FF et notre implémentation sont toutes motivées par nos expérimentations, puisque trouver des décisions optimales pour ces problèmes d'ordonnancement entre actions n'est pas un calcul polynomial (cf. [8]).
La fonction **calculer_heuristique** retourne une structure $\langle PR, U, S\rangle$ où : PR est un plan relaxé, U est un ensemble d'actions utiles, et S est un ensemble d'actions de secours. Comme la complétude est préservée par l'algorithme, nous pouvons avoir un ensemble d'actions utiles plus réduit que celui de FF.

calculer_anticipation(E, PR) : cette fonction recherche un plan anticipé valide pour un état E, en utilisant les actions du plan relaxé PR retourné par **calculer_heuristique** (cf. Figure 2). Plusieurs stratégies peuvent être imaginées : rechercher des plans avec un nombre limité d'actions, retourner plusieurs plans anticipés, ... La meilleure stratégie que nous ayons trouvée consiste à rechercher un seul plan anticipé, contenant le plus possible d'actions du plan relaxé, et d'essayer de remplacer une action de PR qui n'est pas applicable par une autre action (choisie dans l'ensemble des actions du problème) quand aucun autre choix n'est possible.

L'algorithme fonctionne de la manière suivante. On entre tout d'abord dans une boucle principale qui s'arrête lorsqu'aucune action n'a pu être ajoutée au plan anticipé en cours de construction, ou lorsque toutes les actions du plan relaxé PR ont été utilisées. Dans cette boucle, on trouve deux parties : la première permet de sélectionner des actions du plan relaxé, et la deuxième cherche à remplacer une action de PR par une autre action (appartenant à l'ensemble des actions du problème) lorsque la première partie a échoué (on n'a pas trouvé d'actions de PR applicables dans E).

Dans la première partie, les actions de PR sont observées l'une après l'autre, dans l'ordre dans lequel elles apparaissent dans la séquence. Chaque fois qu'une action a est applicable à E, a est ajoutée à la fin du plan anticipé. E est mis à jour en lui appliquant a. Les actions qui ne peuvent pas être appliquées sont collectées dans un nouveau plan relaxé (variable $\acute{e}chec$), dans l'ordre dans lequel dans lequel elles sont sélectionnées. Si au moins une action de PR est applicable (et est donc ajoutée au plan anticipé), quand toutes les actions de PR auront été essayées, la seconde partie ne sera pas utilisée (ceci est contrôlé par le booléen $continuer$). Le plan relaxé est enfin mis à jour avec $\acute{e}chec$, et ce procédé se répète jusqu'à ce que PR soit vide ou qu'aucune action ne puisse être ajoutée au plan anticipé.

La seconde partie est déclenchée quand aucune action n'a pu être ajoutée au plan anticipé dans la première partie. Le but est d'essayer de réparer le plan relaxé courant (dont aucune action n'est applicable), en remplaçant une de ses actions par une action appartenant à l'ensemble des actions du problème, applicable dans l'état anticipé courant E. Les actions de $\acute{e}chec$ sont observées l'une après l'autre, et on sélectionne une action de l'ensemble des actions du problème applicable dans E, produisant au moins un ajout de l'action de $\acute{e}chec$ que l'on observe, cet ajout étant en outre une précondition d'une action de PR n'appartenant pas à E. Si plusieurs actions sont possibles pour le même ajout de l'action de $\acute{e}chec$ que l'on observe, on sélectionne celle qui a le coût minimum dans le graphe relaxé utilisé pour l'extraction du plan relaxé (fonction **choisir_meilleur**). Quand une telle action est trouvée, elle est ajoutée au plan anticipé et la boucle principale est répétée. L'action de $\acute{e}chec$ observée quand on a trouvé une action de remplacement n'est pas conservée dans le plan relaxé.

5 Évaluation expérimentale

5.1 Planificateurs, benchmarks et objectifs

Nous comparons quatre planificateurs : FF v2.3[3] implémenté en C, et trois différentes versions de notre système de planification appelé YAHSP (pour "Yet Another Heuristic Search Planner"[4]) implémenté en Objective Caml[5] compilé pour la rapidité :

[3] http ://www.informatik.uni-freiburg.de/∼hoffmann/ff.html

```
fonction calcul-anticipation (E, PR)    /* E : état, PR : plan relaxé */
  soit plan = ⟨⟩ ;    /* le plan anticipé */
  soit échec = ⟨⟩ ;    /* le prochain plan relaxé */
  soit continuer = vrai ;
  tantque continuer ∧ PR ≠ ⟨⟩ faire
    continuer ← faux ;
    pour i ∈ [1, n] faire    /* avec PR = ⟨a_1, ..., a_n⟩ */
      si Prec(a_i) ⊆ E alors
        continuer ← vrai ;
        E ← E ↑ a_i ;
        plan ← plan ⊕ ⟨a_i⟩
      sinon
        échec ← échec ⊕ ⟨a_i⟩
      finsi
    finpour ;
    si continuer alors
      PR ← échec ;
      échec ← ⟨⟩
    sinon
      PR ← ⟨⟩ ;
      tantque ¬continuer ∧ échec ≠ ⟨⟩ faire
        pour f ∈ Add(Premier(échec)) faire
          si f ∉ E ∧ ∃a ∈ (PR ⊕ échec) | f ∈ Prec(a) alors
            soit AP = {a ∈ A | f ∈ Add(a) ∧ Prec(a) ⊆ E} ;
                /* actions possibles */
            si AP ≠ ∅ alors
              soit a = choisir_meilleur(AP) ;
              continuer ← vrai ;
              E ← E ↑ a ;
              plan ← plan ⊕ ⟨a⟩ ;
              PR ← PR ⊕ Reste(échec) ;
              échec ← ⟨⟩
            finsi
          finsi
        finpour ;
        si ¬continuer alors
          PR ← PR ⊕ ⟨Premier(échec)⟩ ;
          échec ← Reste(échec)
        finsi
      fintantque
    finsi
  fintantque
  retourner(E, plan)
fin
```

FIG. 2 – Algorithme de calcul des états et des plans anticipés

- BFS (Best First Search) : recherche classique de type *WA**, avec *W* = 3. L'heuristique est basée sur le calcul d'un plan relaxé comme pour FF, avec les différences décrites en Section 4.
- OBFS (Optimistic Best First Search) : identique à BFS, en utilisant la distinction entre actions utiles et actions de secours. Un noeud contenant des actions utiles est toujours préféré à un noeud contenant des actions de secours.
- LOBFS (Lookahead Optimistic Best First Search) : identique à OBFS, avec l'utilisation des états et plans

[4] http ://www.cril.univ-artois.fr/∼vidal/yahsp.html

[5] Objective Caml est un langage fonctionnel fortement typé de la famille ML, avec une extension orientée objet (http ://caml.inria.fr/).

anticipés.

Nous utilisons neuf domaines de test[6] : le domaine bien connu Logistics, les domaines Mprime et Mystery créés pour la $1^{\text{ère}}$ IPC, le domaine Freecell créé pour la $2^{\text{ème}}$ IPC, et les cinq domaines STRIPS créés pour la $3^{\text{ème}}$ IPC (Rovers, DriverLog, Satellite, ZenoTravel, Depots). Les problèmes pour Logistics sont ceux de la $2^{\text{ème}}$ IPC et ceux pour Freecell sont ceux de la $3^{\text{ème}}$ IPC.

Nous avons classé ces domaines dans trois catégories, *en fonction de la façon dont LOBFS les résout (ce qui n'est pas forcément lié à la difficulté intrinsèque de résoudre les problèmes de ces domaines, cf. [7])* : problèmes faciles (Logistics, Satellite, DriverLog, Rovers, ZenoTravel), problèmes de difficulté moyenne (Mprime, Freecell), et problèmes difficiles (Depots, Mystery).

Nos objectifs pour ces expérimentations sont les suivants :

1. *Tester l'efficacité de notre planificateur par rapport à un des planificateurs actuels les plus rapides, FF*. En effet, FF est connu pour ses performances remarquables dans de nombreux domaines et ses succès lors des deux dernières compétitions de planificateurs. Bien que généralement moins rapide que FF dans ses versions BFS et OBFS, notre planificateur a des performances proches de celles de FF.
2. *Évaluer la stratégie de recherche "optimiste"*. Cette stratégie nous permet d'utiliser les actions utiles dans un algorithme de recherche complet. Ceci est en contraste avec leur utilisation par une recherche locale dans FF. Nous verrons en particulier que OBFS est meilleur que BFS dans quasiment tous les problèmes.
3. *Démontrer que l'utilisation d'une stratégie d'anticipation améliore fortement les performances de la planification par recherche heuristique pour de nombreux problèmes*. De plus, nous verrons que notre planificateur dans sa version LOBFS peut résoudre des problèmes substantiellement plus important que les autre planificateurs (jusqu'à 10 fois plus d'atomes dans l'état initial, et 16 fois plus de buts pour le dernier problème du domaine DriverLog).

Tous les tests ont été effectués sur un Pentium II - 450MHz avec 512Mo de mémoire sous Debian GNU/Linux 3.0. Le temps limite pour la résolution de chaque problème a été fixé à une heure. Dans tous les graphiques, les problèmes sont ordonnés suivant les valeurs croissantes du temps de calcul pour LOBFS.

5.2 Problèmes "faciles"

Comme les problèmes originaux des différentes compétitions sont résolus très facilement par LOBFS, nous avons créé 10 problèmes supplémentaires dans chaque domaine. Les problèmes numérotés de 1 à 20 sont les problèmes originaux, ceux numérotés de 21 à 30 sont les problèmes nouvellement créés.

Afin de bien comprendre la portée des résultats présentés ici, il est important de remarquer ceci : *la difficulté (mesurée par le nombre d'atomes présents dans l'état initial et le nombre de buts) entre chacun des nouveaux problèmes augmente beaucoup plus rapidement qu'entre les problèmes originaux.* En effet, le dernier problème que nous avons créé pour chaque domaine est le plus large pouvant être résolu par LOBFS par rapport à la mémoire vive disponible. Comme le temps de calcul reste raisonnable, des problèmes plus importants pourraient sûrement être résolus en moins d'une heure avec plus de mémoire.

Par conséquent, le graphe qui représente la longueur des plans obtenus est divisé en deux parties : la longueur des plans pour les problèmes que nous avons créés augmente beaucoup plus rapidement que pour les problèmes originaux. Nous montrons aussi dans la Table 1 quelques données à propos des problèmes les plus larges résolus par FF, OBFS et LOBFS, afin de réaliser les progrès accomplis dans la taille des problèmes pouvant être traités par un planificateur.

Pour les cinq domaines présentés dans cette section, la supériorité de LOBFS sur tous les autres planificateurs ne fait aucun doute. De même, OBFS est clairement supérieur à BFS, tandis que OBFS et FF ont des performances comparables (voir Figure 3). Des résultats similaires sont observés dans de nombreux autres domaines (Ferry, Gripper, Miconic-10 elevator...). Une raison de l'efficacité dans ces domaines est le très faible nombre de noeuds développés, comme on peut le constater dans la Table 1.

Le seul inconvénient de LOBFS est parfois une légère dégradation de la qualité du plan, mais ceci reste limité : le compromis entre rapidité et qualité penche très nettement en faveur de notre technique d'anticipation.

5.3 Problèmes de "difficulté moyenne"

Bien que moins impressionnantes que pour les quatre premiers domaines que nous avons étudiés, les améliorations obtenues en utilisant la technique d'anticipation est toujours intéressante pour ces deux domaines puisque LOBFS a de bien meilleures performances que OBFS (voir la Figure 4). Ce dernier se comporte bien par rapport à FF, et est plus efficace que BFS. La perte de qualité observée pour LOBFS reste limitée à un petit nombre d'actions, et par exemple dans le domaine Mprime où LOBFS résout tout les problèmes en moins de 10 secondes, nous pourrions utiliser LOBFS pour obtenir une solution très rapidement et ensuite utiliser un autre réglage pour obtenir une solution meilleure.

5.4 Problèmes "difficiles"

A cause de la perte en qualité du plan, l'utilisation des techniques d'anticipation est moins intéressantes que dans les domaines précédents. Elle permettent cependant de trouver des plans à des problèmes pour lesquels OBFS échoue, et d'améliorer le temps de calcul pour plusieurs problèmes (voir la Figure 5). Des développements futurs des idées présentées dans cet article pourraient viser à améliorer le

[6]Tous les domaines et problèmes utilisés dans nos expérimentations sont téléchargeables sur la page web de YAHSP.

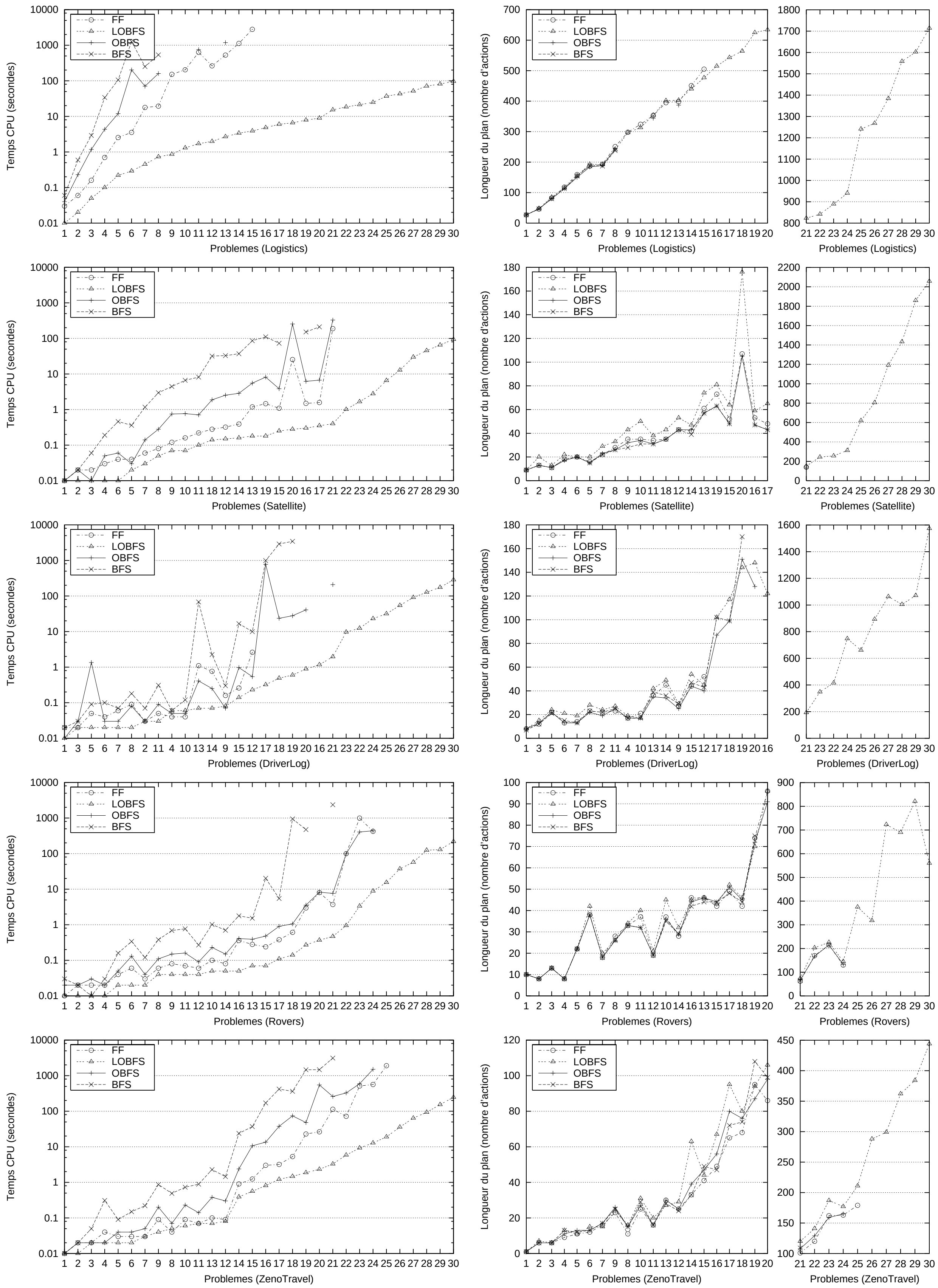

FIG. 3 – Domaines "faciles"

	Logistics 13			Logistics 15		Logistics 30	Satellite 21			Satellite 30
Fluents init	320			364		1140	971			10374
Buts	65			75		200	124			768
	FF	OBFS	LOBFS	FF	LOBFS	LOBFS	FF	OBFS	LOBFS	LOBFS
Long. plan	398	387	403	505	477	1714	140	125	151	2058
Noeuds dév.	16456	16456	4	45785	4	5	22385	20370	5	5
Recherche (s)	527,21	1181,95	0,29	2792,51	0,42	16,64	188,69	328,42	0,12	33,73
Temps total (s)	528,10	1184,35	2,68	2793,82	3,88	96,69	188,82	328,70	0,40	94,24
	Driver 15			Driver 21		Driver 30	Rovers 24			Rovers 30
Fluents init	227			607		2130	5920			35791
Buts	10			38		163	33			127
	FF	OBFS	LOBFS	FF	LOBFS	LOBFS	FF	OBFS	LOBFS	LOBFS
Long. plan	44	44	54	184	193	1574	130	133	145	560
Noeuds dév.	161	273	4	3266	8	38	3876	2114	9	24
Recherche (s)	0,21	0,84	0,02	207,89	0,45	93,92	418,32	430,95	1,97	44,35
Temps total (s)	0,26	0,97	0,14	209,40	1,96	284,65	422,48	437,92	8,87	219,13
	Zeno 24			Zeno 25		Zeno 30				
Fluents init	166			183		353				
Buts	45			49		100				
	FF	OBFS	LOBFS	FF	LOBFS	LOBFS				
Long. plan	163	165	177	179	211	444				
Noeuds dév.	3481	5271	15	8714	16	20				
Recherche (s)	562,09	1496,81	4,15	1898,26	6,45	59,67				
Temps total (s)	564,07	1505,43	12,80	1901,03	18,98	247,06				

TAB. 1 – Problèmes les plus larges dans les domaines “faciles”

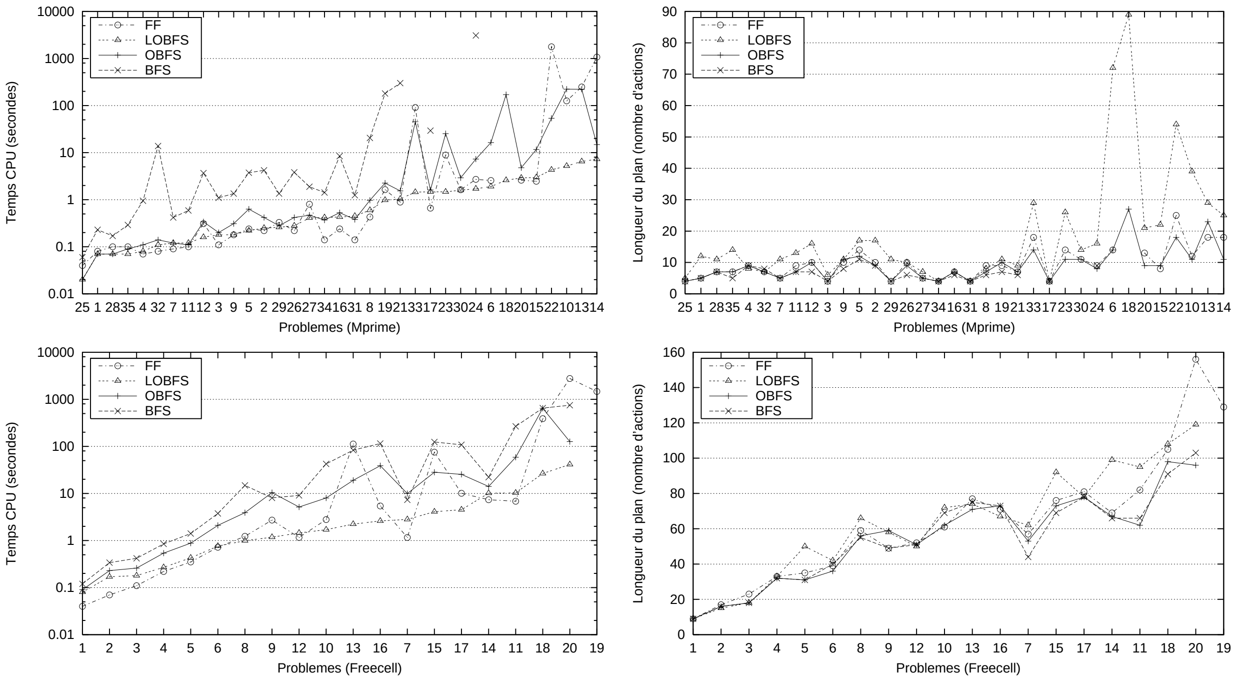

FIG. 4 – Domaines de “difficulté moyenne”

comportement de LOBFS dans ce type de problème, pour lesquels il y a beaucoup d'interactions entre fluents comme le domaine Depots, ou des ressources limitées comme le domaine Mystery.

6 Conclusion

Nous avons présenté une nouvelle méthode pour extraire de l'information d'un plan relaxé, par le calcul de plans anticipés. Ils sont utilisés dans un algorithme complet de type meilleur-d'abord pour calculer de nouveaux noeuds qui peuvent nous rapprocher de la solution. Nous avons amélioré cet algorithme par l'utilisation des actions utiles d'une façon différente de FF, qui préserve la complétude de l'algorithme de recherche. Bien que les états anticipés ne soient généralement pas des états buts et que le facteur de branchement soit augmenté pour chaque état anticipé,

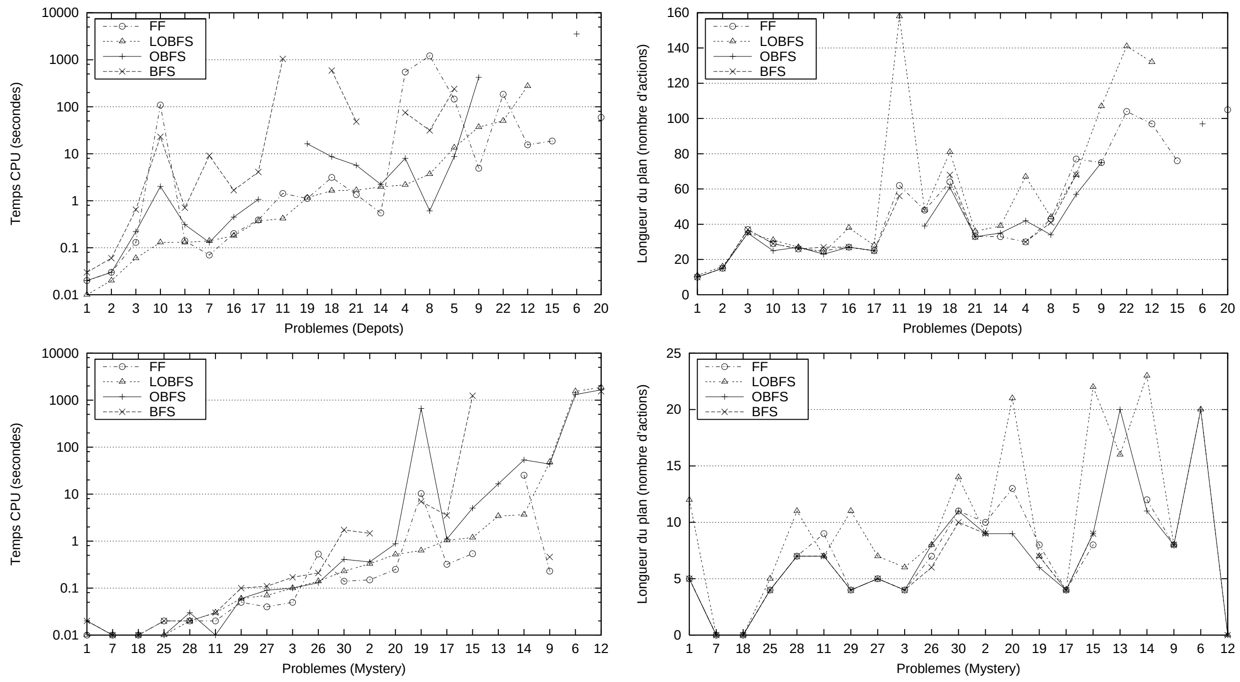

FIG. 5 – Domaines “difficiles”

nos expérimentations prouvent que dans de nombreux domaines (Rovers, Logistics, DriverLog…), notre planificateur est capable de résoudre des problèmes comportant jusqu'à 10 fois plus de fluents dans l'état initial et dans le but que ceux résolus par FF ou par l'algorithme de recherche meilleur-d'abord optimiste sans anticipation. L'efficacité pour les problèmes résolus par tous les planificateurs testés est aussi fortement améliorée avec cette technique d'anticipation. Dans les domaines qui présentent plus de difficulté pour tous les planificateurs (Mystery, Depots), l'utilisation de l'anticipation améliore encore souvent le temps de calcul. La contrepartie de telles améliorations du temps de calcul et de la taille des problèmes pouvant être résolus réside dans une légère perte de qualité de la solution, qui peut dans certains cas être relativement importante. Cependant, cela arrive rarement et la qualité des plans obtenus est en général comparable à celle des plans trouvés par FF.

Remerciements

Merci beaucoup à Pierre Régnier pour ses relectures et les nombreuses discussions…

Références

[1] A. Blum and M. Furst. Fast planning through planning-graphs analysis. Dans *Proc. IJCAI-95*, pages 1636–1642, 1995.

[2] A. Blum and M. Furst. Fast planning through planning-graphs analysis. *Artificial Intelligence*, vol. 90, n° 1-2, pages 281–300, 1997.

[3] B. Bonet and H. Geffner. Planning as heuristic search : New results. Dans *Proc. ECP-99*, pages 360–372, 1999.

[4] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, vol. 129, n° 1-2, pages 5–33, 2001.

[5] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. Dans *Proc. AAAI-97*, pages 714–719, 1997.

[6] R.E. Fikes and N.J. Nilsson. STRIPS : a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, vol. 2, n° 3-4, pages 189–208, 1971.

[7] M. Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, vol. 143, n° 2, pages 219–262, 2003.

[8] J. Hoffmann and B. Nebel. The FF planning system : Fast plan generation through heuristic search. *JAIR*, vol. 14, pages 253–302, 2001.

[9] H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. Dans *Proc. IJCAI-99*, pages 318–325, 1999.

[10] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning-graphs to an ADL subset. Dans *Proc. ECP-97*, pages 273–285, 1997.

[11] D. Long and M. Fox. The efficient implementation of the plan-graph in STAN. *JAIR*, vol. 10, pages 87–115, 1999.

[12] X.L. Nguyen, S. Kambhampati, and R.S. Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, vol. 135, n° 1-2, pages 73–123, 2002.

[13] J. Pearl. *Heuristics*. Morgan Kaufmann, San Mateo, CA, 1983.

[14] D.S. Weld, C.R. Anderson, and D.E. Smith. Extending Graphplan to handle uncertainty and sensing actions. Dans *Proc. AAAI-98*, pages 897–904, 1998.