Chapitre 4 Stratégies de Recherche pour les SP

0. Introduction

Nous allons donner quelques stratégies de contrôle pour les systèmes de production (SP) dont le rôle est de sélectionner la règle à déclencher dans l'étape de sélection de la règle dans la procédure vue dans le chapitre des SP. Pour les SP décomposables, elle aura une tache supplémentaire qui est la sélection de la BDG composante ainsi que la règle composante à déclencher. D'autres taches supplémentaires peuvent lui être affectées:

- Vérifier les conditions d'applicabilité des règles,
- Test de terminaison,
- Mémoriser les règles déjà appliquées, etc…

Une caractéristique de la sélection est la quantité d'informations (de connaissances) sur le domaine pour faire la meilleure sélection (meilleure règle). La sélection de la règle à appliquer peut être:

- arbitraire: Recherche aveugle (non informée). C'est une recherche sans tenir compte d'informations disponibles sur le problème.
- Stratégie guidée : on utilise des connaissances disponibles sur le problème pour guider le choix de la bonne règle (recherche informée).

On peut donc affirmer que l'efficacité d'un SP peut dépendre du:

- Coût d'application des règles (nombre de règles à appliquer),
- Coût du contrôle (effort fourni pour avoir une stratégie informée).

- Avec une stratégie aveugle, le dernier coût peut être modeste (car on ne fournit aucun effort pour pouvoir faire des choix de la règle, le choix est au hasard), mais elle entraînera un coût élevé pour le 1er coût (il faut essayer en général beaucoup de règles avant d'atteindre le but).

- Informer complètement un système sur le domaine nécessite une stratégie de contrôle coûteuse pour avoir cette information, en revanche on minimise le coût d'application des règles car on guide le SP lors de chaque choix (la stratégie guide le système vers le but).

On peut dresser les courbes suivantes des 2 coûts précédents et de déduire le coût global du système

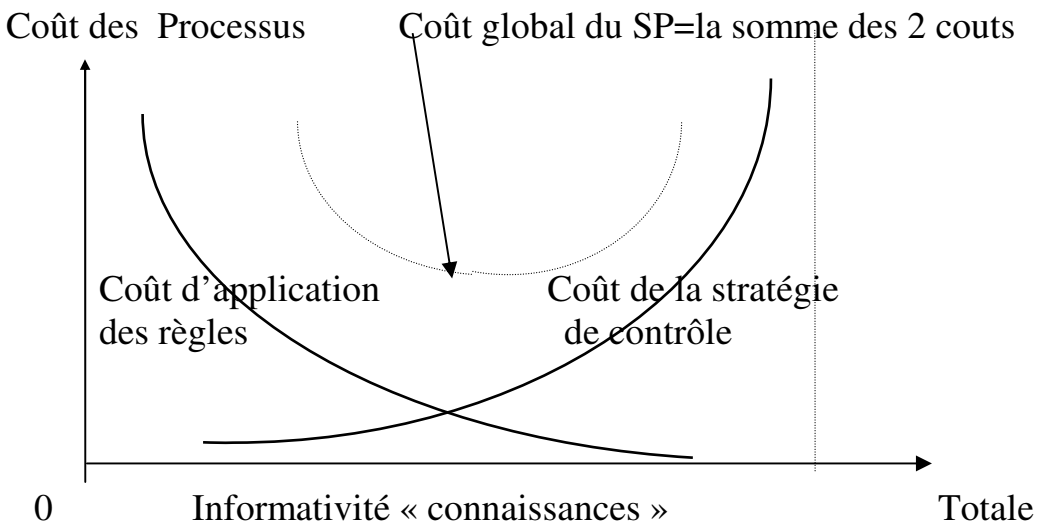

Fig: 1 Coût de calcul des SP

Il serait intéressant de concevoir des SP où on équilibre les 2 coûts. L'efficacité optimale pourrait être obtenue en utilisant des stratégies qui ne soient pas complètement informées mais aussi non nullement informée.

On a vu qu'il existe plusieurs stratégies pour les SP:

- Irrévocable (pour les systèmes commutatifs)
- Par essai successifs:
  - * Retour Arrière (RA) chronologique (simple à implémenter et nécessitant peu d'espace mémoire)
  - * Recherche avec graphe.

On s'intéressera à la 2eme catégorie des stratégies (par essai successifs) (1ere catégorie de stratégie n'est utilisée que pour les SP commutatifs).

1. Stratégie de Retour Arrière:

On donne une procédure récursive **Retour_arrière** qui a en entrée un argument « Donnée » contenant la BDG initiale et retourne la séquence de règles à appliquer pour atteindre le but ou ECHEC si cette séquence n'existe pas.

Procédure Récursive Retour_arierre(Donnée)
Début
1) ***Si*** *Terminé(**Donnée**)* */* Donnée satisfait la condition d'arrêt */*
Alors *renvoyer(**Nil**)*
2) ***Si*** *Impasse(**Donnée**)* */* Donnée n'est pas sur le bon chemin */*
Alors *renvoyer(**échec**)*
3) ***Règles*** *← app_regles(**Donnée**) /* une fonction qui calcule les règles */*
*/*applicables à Donnée et les ordonne arbitrairement ou suivant une heuristique */*
4) Boucle: ***si*** *vide(**Règles**)*
Alors *renvoyer(**échec**) /* plus de règles à appliquer */*
5) ***R*** *← première(**Règles**) /* Meilleure règle est sélectionnée */*
6) ***Règles*** *← queue(**Règles**) /* On enlève la règle sélectionnée */*
7) ***RDonnée*** *← R(**Donnée**) /* produire une nouvelle BDG en appliquant R*/*
*8) Chemin ← retour_arriere(**RDonnée**) /*Appel récursif de la procédure */*
9) ***Si*** *Chemin =* ***Echec*** */*si chemin retournée par l'appel récursif est ECHEC */*
alors *Aller à Boucle /* aller essayer une autre règle */*
10) Renvoyer (R.Chemin) / Concaténation de la règle R au Chemin */*
Fin

Remarques: On peut faire les remarques suivantes sur cette procédure:

- Elle ne se termine avec succès (non échec) que lorsqu'elle produit une BDG satisfaisant le But.
- La liste des règles à appliquer est construite en ligne 10
- Lorsqu'un échec se produit dans un appel récursif, on fait un Retour arrière au dernier point de choix (aller à boucle).
- La ligne 2 accomplit un test pour voir si on est sur le bon chemin.
- En phase 3, les règles applicables sont ordonnées. A ce niveau toute connaissance peut être utilisée. Par définition, si les règles correctes (on entend par règles correctes : les meilleures règles) sont choisies, il n'y aura aucun retour arrière. Par contre si le choix de la règle est arbitraire, la procédure devient inefficace.

Problème: En plus de ces remarques, on peut observer le problème suivant: La procédure peut ne jamais terminer, elle peut:

- Produire de nouvelles BDG indéfiniment,
- Boucler sur une même BDG.

Une solution envisageable à ce problème peut être la suivante:

- Imposer une limite de profondeur des appels récursifs
- Mémoriser les BD déjà produites pour éviter de boucler sur la meme Base.

Il faut donc :

- ajouter une variable globale « limite » qui limite la profondeur (le nombre d'appels récursifs).
- que tout le chemin soit un argument de la procédure.

RA Intelligent : Dans la procédure précédente, lorsqu'on effectue un retour arrière, on le fait au niveau juste inférieur; il existe d'autres systèmes intelligents pour faire des retours arrière directement au bon niveau car en général le meilleur niveau se trouve à un niveau inférieur. C'est ce qu'on appelle un R.A. intelligent.

Dans notre procédure précédente, on ne se souvient pas de toutes les BDG visitées mais uniquement celles se trouvant sur le chemin de l'état initial à l'état courant. Le retour arrière implique que toutes les BDG aboutissant à des échecs sont oubliées. Pour remédier à cet oubli, d'autres stratégies appelées stratégies de recherche avec graphe existent.

2. Stratégie de recherche avec graphe

Dans cette stratégie, on suppose que les BDG produites sont représentées comme les noeuds d'un graphe ou d'un arbre et les arcs correspondent aux règles. L'inconvénient est qu'on obtient un graphe volumineux. Une solution est de ne conserver que la BDG initiale et les traces des changements incrémentaux à partir desquels toute autre BDG peut être obtenue. Une stratégie de recherche avec graphe peut être considérée comme un moyen de trouver un chemin qui relie la BDG initiale à un noeud représentant le but.

2.1 Introduction: Un graphe peut être décrit :

- explicitement en donnant sous forme d'un tableau, les noeuds du graphe (les BDG) avec les arcs (règles) ainsi que les couts associés (méthode inutilisable pour les graphes de taille importante).
- implicitement en donnant le noeud de départ (la BDG initiale) et une fonction de succession qui est appliquée à tout noeud pour donner ses successeurs avec les coûts associés. On introduit un opérateur de succession qui appliqué à un noeud donne tous ses successeurs avec leur couts associés (c'est le développement du nœud).

Le problème dans les SP avec une telle stratégie consiste donc à trouver une séquence de règles c-à-d trouver un chemin de la BDG initiale à la BDG satisfaisant le But avec des fois une contrainte que le chemin soit de coût minimal.

2.2 Une procédure générale de recherche avec graphe

Une procédure qui consiste à développer une partie d'un graphe défini implicitement, peut être définie comme suit:

Procédure recherche_avec_graphe;
1)Créer un graphe de recherche G qui consiste uniquement en noeud départ d. Mettre d sur une liste appelée OUVERT.
2) Créer une liste appelée FERME qui est initialement vide.
*3) BOUCLE: **SI** OUVERT est vide **alors** ECHEC*
4) Sélectionner le 1er noeud de OUVERT, l'enlever de OUVERT et le mettre dans FERME, appeler ce noeud n.
*5) **Si** n est un noeud But*
__alors__ Terminer avec succès. Renvoyer le chemin obtenu le long des pointeurs (construits en phase 7) de n jusqu'à d dans G.
6) Développer le noeud n, produisant l'ensemble M de ses successeurs et les mémoriser comme successeurs de n.
7) Mémoriser un pointeur vers n à partir des éléments de M. Ajouter ces éléments à OUVERT
8)Réordonner OUVERT, soit arbitrairement soit selon une heuristique
9) Aller à Boucle.
Fin

Cette procédure est assez générale pour produire une famille d'algorithmes de recherche avec graphe, la différence entre ces algorithmes réside dans la manière de réordonner les noeuds. L'arbre de recherche est défini par ses pointeurs qui sont établis dans l'étape 7. Chaque noeud excepté d a un pointeur orienté vers un seul de ses parents dans G, qui définit son parent unique. Les noeuds de OUVERT sont les noeuds feuilles de l'arbre de recherche (qui n'ont pas encore été sélectionnés pour etre développés) et les noeuds de FERME sont les autres nœuds (soit des feuilles qui n'ont pas produits de successeurs soit les nœuds qui ne sont pas des feuilles).

Remarques: Dans cette procédure les noeuds sont ordonnés en phase 8 pour sélectionner les meilleures pour être développées en phase 4. Cet ordre peut être arbitraire ou suivant des heuristiques.

- Chaque fois que le noeud sélectionné peut être développé en un noeud but, le processus se termine avec succès, auquel cas le chemin inverse est parcouru de l'état courant jusqu'à d suivant les pointeurs de la phase 4.
- Au contraire un échec peut arriver lorsque le (les) noeud(s) but(s) sont inaccessibles à partir du noeud de départ (liste OUVERT devient vide sans atteindre le but).

3. Procédures aveugles de recherche avec graphe

Si aucune information sur le domaine n'est disponible dans le classement des noeuds dans « ouvert », un plan arbitraire doit être utilisé à la phase 8 de l'algorithme: une telle procédure est dite aveugle (non informée).

3.1 Recherche en profondeur d'abord: Dans ce type de recherche aveugle, les noeuds de « Ouvert » sont ordonnés en ordre **décroissant** suivant leur profondeur dans l'arbre de recherche. Les plus profonds sont placés en 1er. Ceux de même profondeur sont classés arbitrairement. On impose une limite de profondeur pour empêcher que le système ne s'égare sur des chemins infinis. Ce type de recherche produit de nouvelles bases dans un ordre similaire à celui qui est produit par un mécanisme de contrôle utilisant le retour arrière (backtracking). En général on préfère le retour arrière de la recherche en profondeur d'abord car il est plus facile à implémenter et nécessite moins d'espace mémoire, car il ne mémorise pas tous les noeuds produits mais uniquement le chemin menant au but.

3.2 Recherche en Largeur d'abord: Ce type de procédure aveugle ordonne les noeuds dans Ouvert dans l'ordre **croissant** de leur profondeur dans l'arbre de recherche. Il est montré que la recherche en largeur d'abord garantit la découverte du chemin le plus court si celui-ci existe. Dans le cas où il n'existe pas la méthode échouera pour les graphes finis et boucle pour les graphes infinis.

4. Procédure Heuristique de recherche avec graphe

Les méthodes de recherche aveugle sont des méthodes exhaustives visant à découvrir le chemin qui mène à un noeud but. Ces méthodes, malgré qu'elles arrivent à trouver un tel chemin s'il existe, ne sont pas utilisées en pratique car elles développent trop de noeuds avant d'arriver au but. Il faut trouver des alternatives plus efficaces que la recherche aveugle.

On peut utiliser certaines informations dépendant au domaine appelées « informations heuristiques ». Les procédures de recherche utilisant de telles informations sont appelées « procédures heuristiques ». Certaines heuristiques réduisent considérablement l'effort de recherche mais ne garantissent pas de trouver le chemin de moindre coût. En pratique on essaye de minimiser une combinaison du coût du chemin et du coût de la recherche requise pour obtenir ce chemin. On s'intéresse à des méthodes qui minimisent la moyenne de cette combinaison.

On dit qu'une méthode de recherche 1 a plus de puissance heuristique qu'une méthode 2, si la combinaison moyenne du coût de recherche de la méthode 1 est plus basse que celle de la méthode 2.

Les coûts moyens ne sont jamais en réalité calculés, car il est difficile de décider du moyen de combiner le coût du chemin et le coût de recherche et aussi il est difficile de définir une distribution statistique sur le problème. Pour cette raison c'est à l'intuition et l'expérience qu'il appartient de décider si une méthode a plus de puissance heuristique qu'une autre.

4.1. Utilisation des fonctions d'évaluation.

L'information heuristique peut être utilisée pour ordonner les noeuds dans « Ouvert » à l'étape 8 de la procédure précédente. Pour pouvoir classer ces noeuds on peut utiliser une fonction à valeurs réelles sur les noeuds appelée *fonction d'évaluation- (promesse d'un nœud)* (par exemple la probabilité que le noeud se trouve sur le bon chemin).

Soit une telle fonction désignée par f et soit f(n) sa valeur au niveau du noeud n. On supposera que cette fonction donne l'estimation du coût d'un chemin optimal allant du noeud de départ d au noeud but en passant par n. Une fois cette fonction définie, on ordonne les noeuds, se trouvant dans

Ouvert à la phase 8 de la procédure, suivant un ordre croissant (si 2 noeuds ont la même valeur, leur classement est arbitraire en accordant la priorité au noeud but). On suppose qu'un noeud ayant une évaluation basse a plus de chance d'être sur un chemin optimal.

Exemple: Soit le problème du Taquin à 9 cases. On utilise la fonction d'évaluation: f(n)=p(n) + w(n), où p(n) est la profondeur du noeud n dans l'arbre de recherche et w(n) compte le nombre de cases mal placées dans cette base de donnée associée au noeud n. Par exemple la configuration de départ suivante a une valeur f = 0 + 4 = 4.

2	8	3
1	6	4
7	x	5

Etat Initial

$\Rightarrow$

1	2	3
8	x	4
7	6	5

But

Les résultats avec cette fonction d'évaluation sont donnés par: (exercice)

Si on utilise comme fonction f(n)=p(n), on obtient le processus de recherche en largeur d'abord.

Le choix de la fonction a un effet déterminant sur les résultats de la recherche. En effet, l'utilisation d'une fonction qui ne parvient pas a reconnaître certains noeuds prometteurs peut produire des chemins dont le coût n'est pas minimal. En revanche, une fonction qui surestime les résultats que promettent tous les noeuds produit un nombre trop élevé de noeuds développés (exemple recherche en largeur d'abord).

4.2. Algorithme A

Définition: Définissons la fonction d'évaluation f pour que la valeur f(n), estime la somme du coût du chemin optimal depuis le noeud initial d au noeud n, plus le coût du chemin optimal du noeud n au noeud but (c-a-d f(n) est une estimation du coût du chemin optimal passant par n). Le noeud dans « Ouvert » ayant la plus petite valeur de f est donc le noeud estimé et approprié à etre développé en 1er.

Notations: Soit les fonctions k, h* telles que:

$k(n_i,n_j)$ *donne le coût réel entre 2 noeuds* n_i *et* n_j *relié par un arc*.

$k(n,t_i)$ *donne le coût du chemin optimal du noeud n à un noeud but* t_i

h*(n) = *min(k(n,t_i)) sur les t_i (la fonction h* n'est pas définie pour les noeuds à partir desquels le but n'est pas atteints) C'est le cout minimal de n à un nœud but t_i*

k(d,n) *le coût du chemin optimal du noeud de départ d à n.*

g*(n)=min k(d,n) *pour tous les nœuds n accessibles à partir de d.*

f*(n) =g*(n)+h*(n) la somme du coût réel du chemin optimal du noeud d au noeud n et du coût d'un chemin optimal du noeud n au noeud but.

f(n) est donc le coût d'un chemin optimal commençant en d et passant par n et atteint un but.*

Nous souhaitons que f soit une estimation de f* soit f(n)=g(n)+h(n) ou g est une estimation de g* et h une estimation de h*. Un choix simple pour g(n) peut être le coût du chemin dans l'arbre de recherche de d à n donné en additionnant le coût des arcs traversés de n à d. Pour l'estimation h(n) de h*(n), nous comptons sur l'information heuristique du domaine (par exemple similaire à la fonction w(n) du Pb du Taquin). h est appelée la fonction heuristique.

Nous appelons l'algorithme de **Recherche avec Graphe** utilisant la fonction f(n)=g(n)+h(n), *l'Algorithme A.*

Si h est un minorant de h*(c-a-d h(n) ≤ h*(n) ∀n) alors l'algorithme A trouvera un chemin optimal vers un but. Lorsque l'algorithme A utilise une fonction heuristique h qui est un minorant de h*, nous l'appelons A*.

Admissibilité de A*

Nous disons qu'un algorithme de recherche est admissible si, pour tout graphe, il se termine toujours par un chemin optimal de d à un noeud but (lorsqu'un tel chemin existe) c-a-d que pour tout nœud n à partir duquel un nœud but peut etre atteint, h(n) <= au cout du chemin optimal de n à noeud but.

Résultat 1: la procédure **recherche avec graphe** se termine toujours lorsque le graphe est fini.

Résultat 2: A tout moment avant que A* ne se termine, il existe dans Ouvert un noeud n' qui est sur un chemin optimal allant de d à un noeud but, avec f(n') <= f*(d).

Résultat 3: S'il existe un chemin de d à un noeud but, A* se termine.

Résultat 4:L'algorithme A* est admissible, c-a-d que s'il existe un chemin de d à un noeud but, A* se termine en découvrant un chemin optimal.

Résultat 5: Pour tout noeud n sélectionné pour être développé par A*, f(n)<=f*(d).

Comparaison des algorithmes A*

Résultat 6: Si A1 et A2 sont 2 versions de A* telles que A2 est plus informé que A1 (c-a-d que ∀ noeud n non but, h2(n) > h1(n)), alors lorsque les recherches dans tout graphe ayant un chemin entre d et un noeud but se terminent, chaque noeud développé par A2 est développé par A1. Il s'ensuit que A1 développe au moins autant de noeuds que A2.

La condition de Monotonie :

Résultat 7: Si la condition de monotonie (h est monotone si ∀ ni et nj 2 noeuds tels que nj est successeur de ni, h(ni)-h(nj) <= cout(ni,nj)) est vérifiée, alors A* a déjà trouvé un chemin optimal jusqu'au noeud sélectionné pour être développé. C-a-d si A* sélectionne n pour le développer et si la condition de monotonie est respectée alors g(n)=g*(n).

Résultat 8: Si la condition de monotonie est respectée, alors les valeurs f de la séquence de noeuds développes par A* sont croissantes.

5. La Puissance Heuristique des Fonctions d'Evaluation

La sélection de la fonction heuristique est cruciale pour déterminer la puissance heuristique de l'algorithme A. Choisir h=0 (recherche en largeur d'abord) assure l'admissibilité mais s'avère inefficace. Si on prend h égale à

la borne inférieur de h*, on ne développe que les quelques nœuds compatibles avec le maintien de l'admissibilité.

6. Algorithmes Voisins

Recherche Bidirectionnelle: Les méthodes de recherche vues jusqu'à présent utilisent les règles de productions soit en chaînage Avant, soit en chaînage Arrière. Une possibilité intéressante est de chercher dans les 2 directions simultanément. La recherche avance simultanément à partir du noeud de départ et d'un ensemble de noeuds but. Le processus se termine lorsque (et si) les 2 frontières de la recherche se rencontrent.

Recherche par Etapes: D'après son nom, on désire effectuer une recherche par étape. Supposons que nous ayons un problème qui même avec des heuristiques performantes, on atteint les limites de calcul, au lieu d'abandonner l'opération il serait intéressant d'élaguer le graphe pour libérer l'espace mémoire indispensable afin de pousser la recherche plus en profondeur. Il serait intéressant d'élaguer (débarrasser le graphe des chemins inutiles) puis continuer la recherche pour libérer l'espace mémoire. Même si A* est utilisé à chaque stade et que le processus total se termine en trouvant un chemin, il n'est pas garanti que le chemin soit optimal.

Limitations des successeurs: On peut se débarrasser de tous les successeurs n'ayant pas une valeur de f satisfaisante après qu'ils ont été développes. Cet élagage fait perdre au système sa complétude.