CHAPITRE II
Algorithmes de recherche dans un graphe

Le but de ce chapitre est de donner une vue générale sur les méthodes de recherche dans un graphe et leur lien avec la reconnaissance automatique de la parole. On y présente en particulier l'algorithme A*, que nous allons utiliser par la suite pour l'obtention des N meilleures solutions.

1 Généralités

Un graphe G est un couple (Y,U) tel que :

Y : ensemble d'éléments appelés sommets ou noeuds.

U : ensemble des couples d'éléments de Y, appelés arcs.

A ce graphe G est associé un noeud début "n_0". Les paires de noeuds (n,n') de l'ensemble U sont connectées par des arcs directs. Si un arc est orienté d'un noeud n vers un noeud n', on dit que n' est le successeur (ou fils) de n et le noeud n est dit parent (ou père) de n'.

Un chemin dans $G = (Y,U)$ est une suite de sommets telle que $\forall n,n'$ éléments consécutifs, $(n,n') \in U$. Les premier et dernier sommets sont appelés extrémités.

Dans le cas de la reconnaissance automatique de la parole par modèles de Markov cachés, le graphe G représente l'ensemble des chemins possibles dans le réseau acoustique. Un noeud est défini par une position fixée par la paire (q_i, τ) où q_i est un état du réseau et τ un instant donné. Les arcs sont définis par une règle de progression à l'intérieur du réseau acoustique. La figure suivante (II.1) présente un exemple de ce type de graphe. Dans cet exemple, on utilise un modèle de Markov caché à 4 états et une observation de longueur temporelle égale à 4. La partie (I) de la figure présente le graphe de recherche dans son ensemble. Un graphe de recherche est la partie de l'espace des états qui est explicitée au cours de la recherche du noeud objectif (état final). La partie (II) présente les différentes règles (a, b et c) de progression à

l'intérieur du réseau et la partie (III) est une autre représentation du réseau avec les différents noeuds (rotation de 90° de la partie I de la figure II.1). n_0 est le noeud de départ (état 1 à l'instant t=1), n_F est le noeud final (état 4 à l'instant t=4). La numérotation des noeuds ne reflète pas l'ordre dans lequel les noeuds sont développés.

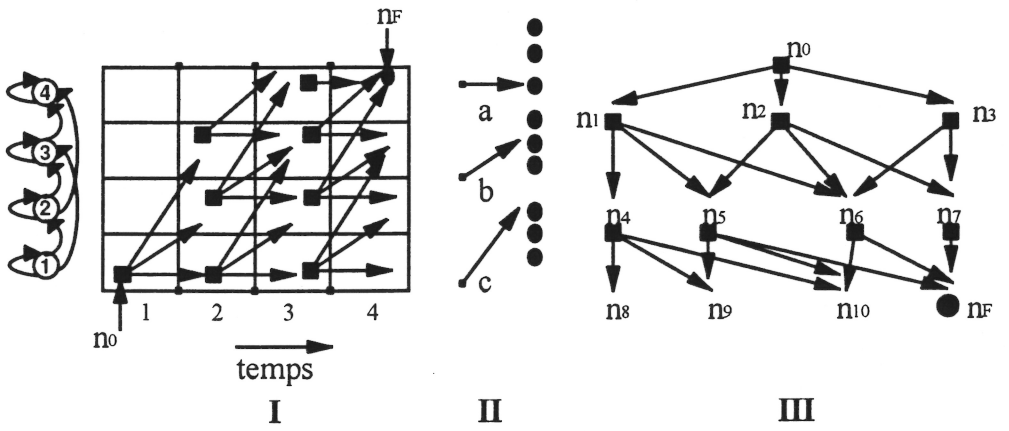


Figure II.1 Exemple d'un graphe de recherche

2 Méthodes de recherche dans un graphe

Une méthode de recherche dans un graphe a pour rôle de déterminer l'ordre dans lequel les noeuds doivent être développés afin de retrouver le noeud objectif (s'il existe) à partir du noeud de départ. Une méthode de recherche sera d'autant "meilleure" qu'elle produira un "petit" ou "peu coûteux" graphe de recherche (par rapport au graphe complet) avec un petit effort de génération[1]. Une solution est obtenue dès l'instant où le noeud objectif apparaîtra dans le graphe de recherche.

On distingue deux catégories de méthodes de recherche utilisant l'une, une recherche aveugle, et l'autre, une recherche ordonnée [Rich, 87], [Pearl, 90].

Dans la méthode de recherche aveugle ou non informée, l'ordre dans lequel les noeuds sont développés ne dépend que de l'information collectée par la recherche mais n'est pas affecté par le caractère inexploré du graphe, ni même par le critère du but. La méthode de recherche ordonnée, informée ou guidée ou dirigée, utilise une information partielle sur le domaine du problème et sur la nature du but pour aider le guidage de la recherche vers une direction plus

[1] Le coût d'une méthode de recherche est fonctions du nombre de noeuds développés et de l'espace mémoire nécessaire pour la recherche.

prometteuse. Nous allons décrire plus en détail ces deux méthodes de recherche dans ce qui suit.

2.1 Méthodes de recherche aveugle

Dans ce type de méthodes, l'algorithme de choix des noeuds à développer ne dépend que des connaissances (estimations ...) qu'on peut avoir de la position du noeud objectif ou d'un chemin vers la position du noeud objectif. Ces méthodes sont : la méthode de recherche en largeur et la méthode de recherche en profondeur.

2.1.1 Recherche en largeur

Cette recherche consiste à générer toute la descendance de la racine en appliquant chacune des règles applicables sur le noeud initial. Ensuite pour chaque noeud fils (de la racine), il faut générer tous ses successeurs en appliquant toutes les règles qui sont appropriées. La progression de recherche est montrée sur la figure (II.2). Les noeuds sont numérotés dans l'ordre de leur développement.

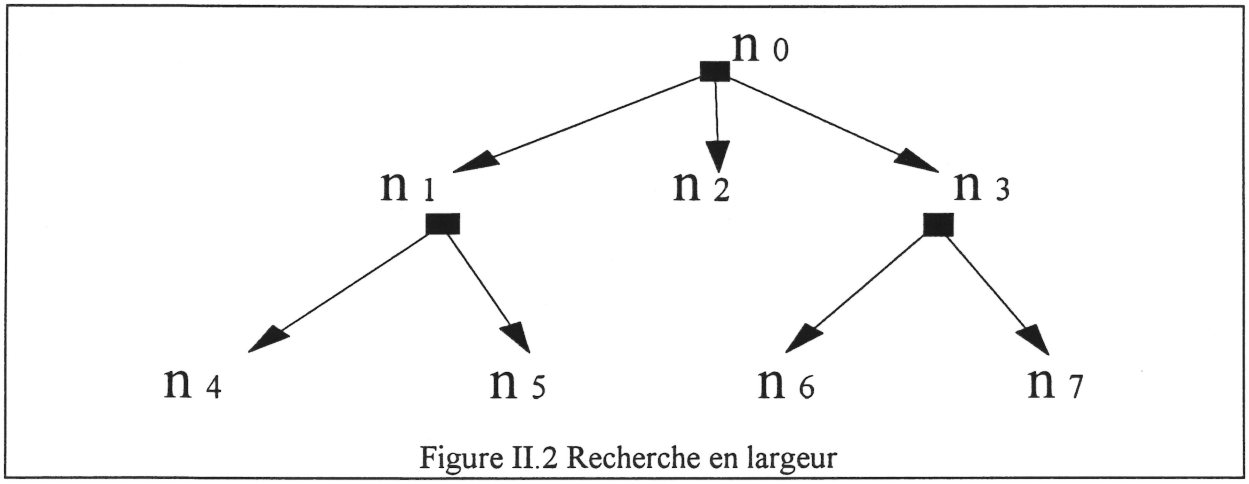


Figure II.2 Recherche en largeur

L'exploration se fait par couche, en utilisant une pile du type FIFO (First In First Out, premier arrivé premier servi). Ces méthodes garantissent l'obtention d'une solution si celle-ci existe et s'il n'y a pas de sommet avec une infinité de successeurs directs (graphe infini). Cette méthode est coûteuse en mémoire.

2.1.2 Recherche en profondeur

Cette méthode consiste à développer tous les successeurs d'un sommet avant de traiter un autre sommet. Lorsqu'il n'y a plus de descendants, on remonte jusqu'au dernier niveau où il reste des sommets inexplorés. Le danger de cette méthode est évidemment d'explorer indéfiniment une branche stérile. C'est pourquoi on prévoit souvent une profondeur limite dans le cas de graphes infinis. La progression de recherche est montrée sur la figure (II.3). Les noeuds sont numérotés dans l'ordre de leur développement.

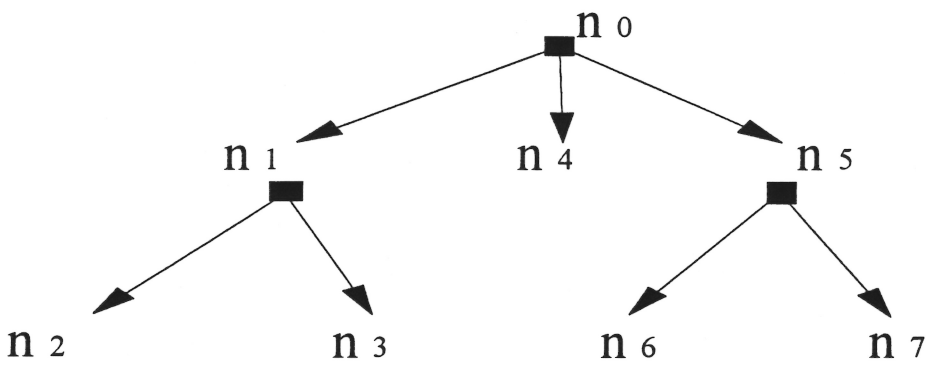


Figure II.3 Recherche en profondeur

Les méthodes en profondeur d'abord correspondent à une gestion de pile en LIFO (Last In First Out, dernier arrivé premier servi).

2.2 Méthode de recherche ordonnée

Les méthodes de recherche aveugles (en profondeur ou en largeur) sont des méthodes exhaustives pour la recherche du noeud final dans un graphe contenant un nombre très élevé de noeuds. Les limitations des méthodes de recherche aveugles sont dues essentiellement au temps de calcul et à la capacité mémoire[1] nécessaire pour le développement de tous les noeuds.

Dans certaines tâches, il est possible d'utiliser d'autres informations disponibles pour guider la recherche et donc pour réduire le temps de traitement. Chaque information est spécifique d'une tâche donnée et d'un graphe donné. Ce type d'information est appelé heuristique.

Une heuristique a pour rôle d'aider à trouver une solution. Elle est comme un guide de voyage. Elle est bonne lorsqu'elle oriente dans la bonne direction, elle est mauvaise quand elle conduit à une impasse.

---

[1] Cette limitation apparaît surtout dans le cas d'une recherche en largeur.

Une première idée d'heuristique pouvant être utilisée dans les méthodes aveugles, consiste à générer les noeuds non plus d'une manière arbitraire mais avec un ordre bien déterminé. Cette stratégie vise à déterminer le meilleur noeud à développer parmi tous les noeuds rencontrés indépendamment de leur position dans le graphe. Cette stratégie est connue sous le nom de l'algorithme du "meilleur d'abord". Elle utilise pour sa recherche une fonction coût f, appliquée à chaque noeud à développer. Elle dépend de la description du noeud n, de la description du but, de l'information recueillie par la recherche jusqu'à ce point et plus important encore, de toutes les connaissances supplémentaires liées au problème. La fonction f représente une estimation du coût minimum global du chemin s, entre la racine et l'objectif, passant par le noeud n.

Plusieurs types d'algorithmes utilisent la stratégie du "meilleur d'abord", parmi eux l'algorithme A*. Les paragraphes suivants vont décrire cet algorithme, ses propriétés et ses différentes caractéristiques.

3 Algorithme A*

Cet algorithme est l'un des plus connus parmi ceux qui sont définis pour un graphe d'états [Hart, 68], [Nilsson, 71], [Hart, 72]. Il est typique de la stratégie du "meilleur d'abord".

Dans la description fournie, on suppose que l'algorithme est utilisé pour la recherche du chemin de coût minimal.

3.1 Fonction coût

La fonction coût f est définie pour chaque noeud n_i. Cette évaluation $f(n_i)$ représente une estimation du coût du chemin (de coût minimum) contraint de passer par le noeud n_i et liant le noeud initial n_0 au noeud final n_F. Elle est obtenue par:

$$f(n_i) = g(n_i) + h(n_i) \qquad \text{(II.1)}$$

où $g(n_i)$ est le coût du chemin (de coût minimum) entre le noeud initial n_0 et le noeud n_i.

et $h(n_i)$ est une estimation du coût du chemin (de coût minimum) entre le noeud n_i et le noeud final n_F.

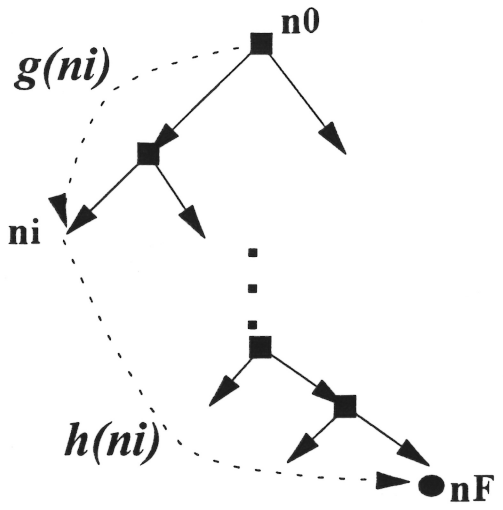


Figure II.4 Les différentes parties de la fonction coût *f*

Lorsque *g* et *h* coïncident avec les valeurs optimales *g** et *h**, la fonction *f* résultante prend une signification particulière, on obtient alors :

$$f^*(n_i) = g^*(n_i) + h^*(n_i) \qquad \text{(II.2)}$$

où $g^*(n_i)$ est le coût minimum du chemin reliant la racine au noeud n_i.

et $h^*(n_i)$ est le coût minimum du chemin reliant le noeud n_i au noeud final n_F.

Nous remarquons que $f^*(n_i)$ mesure alors le coût du chemin optimal contraint à passer par le noeud n_i.

3.2 Stratégie de recherche du noeud final

Pour retrouver le noeud final ou objectif, l'algorithme A* utilise une recherche du type "meilleur d'abord". La fonction coût utilisée a été définie dans le paragraphe (3.1). Dans la procédure de recherche nous distinguons deux sous-ensembles de noeuds :

- Les noeuds qui ont été développés (c'est-à-dire dont les successeurs sont disponibles pour la procédure de recherche) sont appelés "fermés".

- Les noeuds qui ont été générés et qui attendent le développement sont appelés "ouverts".

Deux listes séparées, appelées FERME et OUVERT, sont utilisées pour conserver la trace des ces deux sous-ensembles de noeuds.

L'algorithme de recherche est alors le suivant :

1 Initialisation

Mettre le noeud initial n_0 dans OUVERT.

Calculer $f(n_0) = g(n_0) + h(n_0)$

2. Test d'échec

Si OUVERT est vide sortir avec échec.

3. Choix du noeud à traiter

Retirer de OUVERT et mettre dans FERME le noeud n pour lequel f est minimum.

4. Est-ce la fin ?

Si n est un noeud but, sortir avec la solution obtenue en retraçant (remontant) les pointeurs de n à n_0.

5. Développement

Sinon développer n, pour engendrer tous ses successeurs, et affecter à ces derniers les pointeurs indiquant qu'ils sont successeurs (descendants) de n. Pour tout successeur n' de n :

a. Si n' n'est pas encore dans OUVERT ni dans FERME, faire une estimation de $h(n')$ (estimation du coût du meilleur chemin allant de n' jusqu'à un noeud but), et calculer

$$f(n') = g(n') + h(n')$$

où $g(n') = g(n) + c(n,n')$, avec $c(n,n')$ le coût du chemin entre le noeud n et le noeud n'.

b. Si n' est déjà dans OUVERT ou FERME, modifier éventuellement ses pointeurs pour le rattacher au chemin fournissant $g(n')$ minimum.

c. Si n' a nécessité un ajustement des pointeurs et a été trouvé dans FERME, le remettre dans OUVERT. (Cela signifie que l'on arrive au noeud n' par un nouveau chemin, et qu'il faut reconsidérer les développements de ce noeud).

6. Boucle ...

Aller à l'étape 2.

3.3 Choix de la fonction *h*

L'efficacité de l'algorithme A* est liée à la manière dont la fonction h est estimée. Si h est un estimateur parfait, alors l'algorithme A* convergera immédiatement vers le but sans recherche inutile (sans développement de chemins inutiles). Dans le cas où h est surestimée, on ne peut avoir l'assurance de trouver le chemin solution le moins coûteux. Par contre, si h est sous-estimée, la meilleure solution peut être trouvée. Meilleure est h (c'est-à-dire, plus h est proche de h^* mais inférieur), plus nous obtiendrons rapidement la meilleure solution.

$$h(n_i) \leq h^*(n_i) \tag{II.3}$$

3.4 Admissibilité de l'algorithme A*

1/ Par définition de l'algorithme A*, pour tout noeud final n_F, on a :

$$f(n_F) \geq C \tag{II.4}$$

où C est le coût du chemin de coût minimum.

2/ $\forall$ un chemin optimal P_k, il existe dans OUVERT un noeud n_i appartenant à ce chemin optimal, tel que n_i a été développé par ce chemin optimal.

On va démontrer cette proposition par récurrence.

On suppose que c'est vrai au départ.

Pour le noeud n_i, on développe tous ses successeurs, donc en particulier son successeur sur le chemin optimal P_k. Puisqu'il s'agit d'un chemin optimal, alors ce noeud est forcément dans OUVERT.

3/ Pour le noeud n_i d'un chemin optimal (développé jusqu'à ce noeud n_i) donc vérifiant la propriété 2, on a :

$$f(n_i) = g^*(n_i) + h(n_i) \tag{II.5}$$

Puisque developpé sur le chemin optimal jusqu'au noeud n_i, et donc $\left(g(n_i) = g^*(n_i)\right)$.

Etant donné que $h(n_i) \leq h^*(n_i)$, on a :

$$f(n_i) \leq g^*(n_i) + h^*(n_i) \tag{II.6}$$

soit

$$f(n_i) \leq f^*(n_i) \tag{II.7}$$

ou encore

$$f(n_i) \leq C \tag{II.8}$$

4/ Des propositions 2 et 3 on déduit l'existance dans OUVERT d'un noeud d'un coût inférieur ou égal à C. En particulier lorsque le noeud final n_F est le meilleur (coût minimal) dans OUVERT on a :

$$f(n_F) \leq C \tag{II.9}$$

soit en combinant avec la proposition 1 :

$$f(n_F) = C \tag{II.10}$$

On vient de montrer l'admissibilité du A*. Un algorithme de recherche qui est assuré de trouver un chemin optimal vers un but s'il en existe un, est appelé "admissible" [Gelperin, 77][Martelli, 77] [Nilsson, 80] [Pearl, 90]. Il existe donc toujours un noeud de OUVERT permettant d'avoir un chemin de coût minimum.

A* est admissible si h est minorante et si la taille de la pile est suffisante pour assurer le développement de la solution .

4 Exemple de comparaison

Nous avons cité dans les paragraphes précédents les différentes caractéristiques de chaque méthode de recherche dans un graphe. Nous allons voir maintenant, à travers un exemple, le coût en calcul de chacune de ces méthodes. Le coût d'une méthode est défini comme étant le nombre de noeuds développés et le nombre de noeuds apparus (en attente de développement). On a choisi comme exemple d'illustration le jeu du taquin.

Principe du jeu

Un taquin est un damier 3x3 comportant 8 carreaux numérotés et disposés dans une configuration initiale donnée. L'objectif du taquin est de modifier la position des carreaux pour arriver à une configuration finale choisie préalablement. Les changements de position des carreaux s'effectuent seulement par glissement d'un des carreaux sur un carré vide à partir d'une position adjacente orthogonale. Une recherche du type largeur ou profondeur n'est pas pratique quand le nombre d'états à examiner est très élevé. Une solution est d'introduire une heuristique pour estimer la proximité d'un état avec un état but.

Dans ce type de jeu, on présente deux sortes d'heuristique :

h_1 : dénombrer les carreaux mal placés.

h_2 : évaluer la somme des distances (horizontales et verticales) entre les carreaux non assortis entre deux états. C'est le nombre minimal de déplacements verticaux et horizontaux nécessaires pour ramener chaque carreau au bon endroit.

Le graphe suivant présente un exemple de configurations initiale (à gauche) et finale (à droite) du taquin.

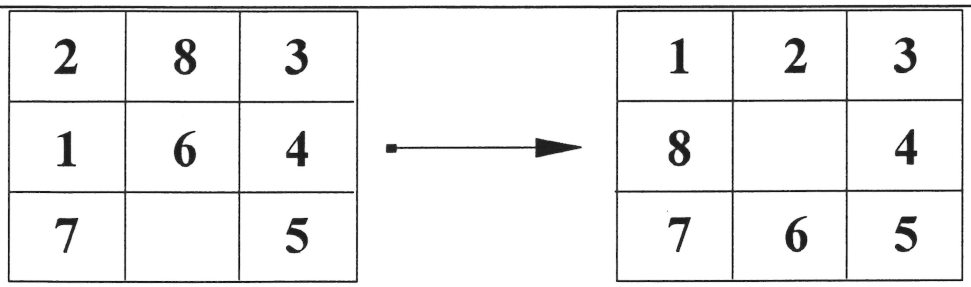


Figure II.5 Etat initial et final du jeu de taquin (exemple)

Les déplacements possibles à partir de la configuration initiale (précédente) sont présentés sur la figure II.6. Pour cette configuration 3 déplacements seulement sont possibles.

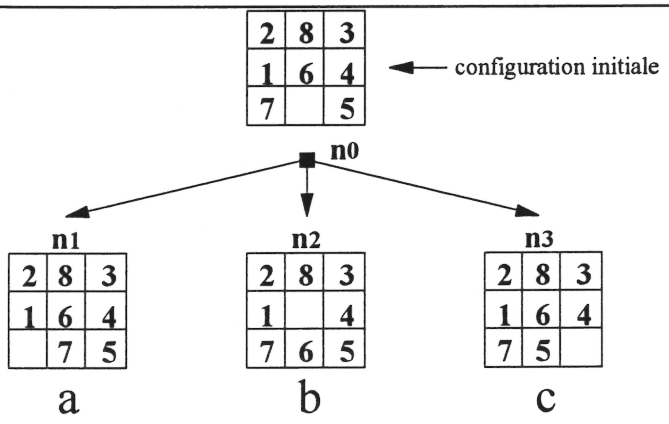


Figure II.6 Graphe de recherche, pour le jeu de taquin

On obtient ainsi à partir du noeud initial n_0 (noeud développé) trois noeuds (n_1 , n_2 et n_3 ; noeuds apparus) représentant les différentes configurations possibles (a, b et c).

Le tableau suivant représente (pour le problème considéré en exemple) les différents coûts d'exploitation obtenus avec les différentes méthodes de recherche utilisées.

	nombre de noeuds développés	nombre de noeuds apparus
En largeur	26	47
En profondeur (niveau 5)	18	35
A* avec heuristique h_1	6	14
A* avec heuristique h_2	5	13

Tableau II.1 Coût de 4 méthodes de recherche du noeud final dans l'exemple du taquin

Les coûts les moins élevés pour retrouver la configuration finale sont obtenus par l'algorithme A*. L'importance de chaque heuristique et elle aussi mise en évidence. La meilleure heuristique est celle qui permet l'obtention la plus rapide du résultat final.

5 Heuristiques dans les techniques de reconnaissance automatique de la parole

Les méthodes de reconnaissance (alignement temporel, méthode statistique ou méthode connexionniste) utilisées sont à base d'algorithmes de recherche dans un graphe. Nous avons présenté dans la première section une représentation graphique de la méthode statistique à base d'algorithmes de programmation dynamique (un alignement temporel entre l'observation et une référence qui n'est autre qu'un modèle de Markov caché). Différentes heuristiques ont été introduites pour réduire l'espace de recherche. Nous allons brièvement en citer quelques unes.

Dans le système de reconnaissance développé par IBM, un modèle de langage est introduit dans la phase de décodage. Ce modèle traduit toutes les phrases admises par l'application et en accepte beaucoup d'autres (c'est le problème des bigrammes et des trigrammes). La recherche de la solution optimale s'effectue par l'algorithme A* (Stack decoding dans [Jelinek, 69]) dans le modèle de langage. Le coût de la fonction f utilisée représente le coût du chemin partiel visité dans le modèle plus une estimation du chemin restant à parcourir pour arriver à la fin de la phrase. L'ensemble des calculs est donné dans [Jelinek, 76]. Dans [Bourlard, 85], on peut trouver une rétrospective complète du système de reconnaissance d'IBM. [Paul, 91] étudia d'autres heuristiques pour estimer le chemin partiel.

Une comparaison entre des méthodes de recherche ordonnées, à bases de plusieurs heuristiques, pour la reconnaissance des mots connectés a été récemment publiée [Ney, 92-b]. Les différents tests effectués montrent l'intérêt d'avoir une bonne heuristique au départ. Quatre méthodes ont été testées : programmation dynamique, recherche heuristique, recherche en largeur et finalement une recherche avec BRANCH-BOUND. Les meilleurs résultats ont été obtenus en utilisant une recherche en faisceau [Ney, 87] avec une programmation dynamique

ou avec une recherche en largeur (il prolongeait tous les chemins possibles). Cependant cette dernière prenait trois fois plus de temps.