Chapitre 1 : Notion de bases sur l’Intelligence Artificielle

1.1 Introduction

L'**intelligence artificielle** (**IA**) est « l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence humaine ». Elle correspond donc à un ensemble de concepts et de technologies plus qu'à une discipline autonome constituée2. Certaines instances.

Cette définition s’accorde sur le fait que l’objectif de l’IA est de créer des systèmes intelligents, mais elles diffèrent significativement dans leur façon de définir l’intelligence. Certaines se focalisent sur le comportement du système, tandis que d’autres considèrent que c’est le fonctionnement interne (le raisonnement) du système qui importe. Une deuxième distinction peut être faite entre celles qui définissent l’intelligence à partir de l’être humain et celles qui ne font pas référence aux humains

mais à un standard de rationalité plus général. On peut donc décliner trois façons de voir l’intelligence artificielle :

1. **Créer des systèmes qui se comportent comme les êtres humains :** cette définition opérationnelle de l’IA fut promue par Alan Turing, qui introduisait son fameux “test de Turing” selon lequel une machine est considérée comme intelligente si elle peut converser de telle manière que les interrogateurs (humains) ne peuvent la distinguer d’un être humain.
2. **Créer des systèmes qui pensent comme des êtres humains** – si l’on adhère à cette deuxième définition, cela implique que l’IA est une science expérimentale, car il faut comprendre au préalable la façon dont pensent les humains (sinon, comment savoir si une machine pense comme un homme ?) et ensuite évaluer les systèmes par rapport à leurs similarités avec le raisonnement humain.
3. **Créer des systèmes qui pensent rationnellement :** selon cette définition, les systèmes doivent raisonner d’une manière rationnelle, c’est à dire en suivant les lois de la logique. Cette approche peut être critiquée car il semble que certaines capacités (la perception, par exemple) ne sont pas facilement exprimables en logique. De plus, ce standard de rationalité ne peut pas être atteint en pratique car la technologie actuelle ne permet pas de réaliser des calculs aussi complexes.

1.2 Une courte histoire de l’IA:

1.2.1 Gestation de l’IA (1943-1955) Pendant cette période furent menés les premiers travaux qui peuvent être considérés comme les débuts de l’intelligence artificielle (même si le terme n’existait pas encore). On peut citer les travaux de McCulloch et Pitts qui ont introduit en 1943 un modèle de neurones artificiels. Quelques années après, Hebb proposa une règle pour modifier des connections entres neurones, et Minsky etEdmonds construisirent le premier réseau de neurones. Ce fut aussi

durant cette période que Turing publia son fameux article dans lequel introduit le test de Turing.

1.2.2 Test de Turing: Le **test de Turing** est une proposition de test d’intelligence artificielle fondée sur la faculté d'une machine à imiter la conversation humaine. Décrit par Alan Turing en 1950 dans sa

publication *Computing Machinery and Intelligence*, ce test consiste à mettre un humain en confrontation verbale à l'aveugle avec un ordinateur et un autre humain.

Si la personne qui engage les conversations n'est pas capable de dire lequel de ses interlocuteurs est un ordinateur, on peut considérer que le logiciel de l'ordinateur a passé avec succès le test. Cela sous-entend que l'ordinateur et l'humain essaieront d'avoir une apparence sémantique humaine.

Pour conserver la simplicité et l'universalité du test, la conversation est limitée à des messages textuels entre les protagonistes.

1.2.3 Naissance d'IA (1956): C'est durant cette année qu'un petit groupe d'informaticiens intéressés par l'étude de l'intelligence se réunirent pour une conférence sur ce thème. Cette conférence dura deux mois (!), et permit de poser les fondements de l'intelligence artificielle (nom qui fut choisi à l'issue de cette conférence)

1.2.4 Espoirs grandissants (1952-1969): Ce fut une période très active pour le jeune domaine de l'IA. Un grande nombre de programmes furent développés pour résoudre des problèmes d'une grande diversité. Les programmes Logic Theorist (par Newell et Simon) et Geometry Theorem Prover (Gelernter) furent en mesure de prouver certains théorèmes mathématiques (tous déjà connus, mais en trouvant parfois une preuve plus élégante). Le General Problem Solver de Newell et Simon réussissait quant à lui à résoudre des puzzles simples avec un raisonnement semblable au raisonnement humain. Samuel créa un programme jouant (à un niveau moyen) aux dames. Des étudiants de Minsky travaillèrent sur les petits problèmes ("microworlds") tels que les problèmes d'analogie (probèmes du même type que ceux des tests de QI), donnant naissance au programme ANALOGY, ou encore les manipulations de cubes (le ameux "blocks world") avec l'idée d'augmenter la complexité petit à petit pour développer des agents intelligents.

1.2.5 Premières Déceptions (1966-1973) Il devint durant ces années de plus en plus évident que les prédictions faites par les chercheurs en lA avaient été beaucoup trop optimistes. Ce fut le cas par exemple pour la traduction automatique. Les chercheurs n'avaient compté que 5 ans pour réaliser un traducteur automatique, mais se sont vite rendu compte que leur approche purement syntaxique n'étaient pas suffisante (pour bien traduire un texte, il faut d'abord le comprendre). Cet échec a provoqué l'annulation en 1966 de tout le financement du gouvernement américain pour les projets de traduction automatique. De grandes déceptions se produisirent également lorsque les chercheurs en IA essayèrent d'appliquer leurs algorithmes aux problèmes de grande taille, et découvrirent alors qu'ils ne fonctionnaient pas, par manque de mémoire et de puissance de calcul. Ce fut une des critiques adressée à l'IA dans le rapport de Lighthill de 1973, qui provoqua l'arrêt du financement de la quasi-totalité des projets en IA de Grande Bretagne. Et comme si cela ne suffisait pas, Minsky et Papert prouvèrent dans leur livre "Perceptrons" de 1969 que les réseaux de neurones de l'époque ne pouvaient pas calculer certaines fonctions pourtant très simples 2 , ce qui mit en cause toute la recherche en apprentissage automatique, entraînant une crise dans cette branche de l'IA.

1.2.6 Systèmes Experts (1969-1979) Le premier système expert, appelé DENDRAL, fut créé en 1969 pour la tâche spécialisée consistant à déterminer la structure moléculaire d'une molécule étant donnés sa formule et les résultats de sa spectrométrie de masse. DENDRAL, comme toutes les systèmes experts, est basé sur un grand nombre de règles heuristiques (nous reviendrons sur ce terme en détail dans la suite du cours) élaborées par des experts humains. Après le succès du DENDRAL, d'autres systèmes d'experts furent créés, notamment le système MYCIN, qui réalisait un

diagnostic des infections sanguines. Avec 450 règles, MYCIN réussissait à diagnostiquer à un niveau proches des experts humains et considérablement meilleur que celui les jeunes médecins.

1.2.7 L'IA Moderne (1987-présent L'intelligence artificielle est devenue au fil du temps une matière scientifique de plus en plus rigoureuse et formelle. La plupart des approches étudiées aujourd'hui sont basées sur des théories mathématiques ou des études expérimentales plutôt que sur l'intuition, et sont appliquées plus souvent aux problèmes issus du monde réel.

1.3 Les sous-domaines de l'IA

On s'en serait douté, créer des agents intelligents n'est pas si simple. Pour cette raison, l'IA s'est divisée en de nombreuses sous-disciplines qui essaient chacune de traiter une partie du problème. Voici les principales :

1.3.1 Répresentation des connaissances et Raisonnement Automatique

Comme son nom le suggère, cette branche de l'IA traite le problème de la représentation des connaissances (qui peuvent être incomplètes, incertaines, ou incohérentes) et de la mise en œuvre du raisonnement.

1.3.2 Résolution de problèmes généraux L'objectif est de créer des algorithmes généraux pour résoudre des problèmes concrets.

1.3.3 Traitement du langage naturel Ce sous-domaine vise à la compréhension, la traduction, ou la production du langage (écrit ou parlé).

1.3.4 Vision artificielle Le but de cette discipline est de permettre aux ordinateurs de comprendre les images et la vidéo (par exemple, de reconnaître des visages ou des chiffres).

1.3.5 Robotique Cette discipline vise à réaliser des agents physiques qui peuvent agir dans le monde (pour voir les robots humanoïdes les plus avancés aujourdhui, allez sur le site http://www.world.honda.com/ASIMO/).

1.3.6 Apprentissage automatique Dans cette branche de l'IA, on essaie de concevoir des programmes qui peuvent s'auto-modifier en fonction de leur expérience.

1.4 L'état de l'art IA

Avant de clore ce chapitre, on peut citer quelques exemples qui illustrent l'état de l'art aujourd'hui :

Jeux En 1997, Deep Blue devient le premier programme à battre un champion du monde d'échecs en titre, ce qui fit sensation. Aujourd'hui, des programmes informatiques peuvent aussi jouer à un niveau expert aux dames, au backgammon, ou encore au bridge. En revanche, le jeu de go s'avère être très résistant, et aucun programme informatique ne dépasse aujourd'hui le niveau d'un joueur de club moyen.

Vision artificielle Le système ALVINN a conduit une voiture à travers des Etats-Unis pendant plus de 4000 kilomètres (avec un peu d'aide pour les moments difficiles comme les sorties d'autoroutes). On peut citer également des programmes de reconnaissance d'écriture qui arrivent à catégoriser les chiffres manuscrits avec moins d'un pourcent d'erreur (voir le site http://yann.lecun.com/exdb/lenet/index.html pour voir quelques démonstrations d'un tel programme).

Planification et Ordonnancement Un système de planification et d'ordonnancement a contrôlé sans la moindre intervention humaine une navette spaciale pendant 2 jours, planifiant ses actions, détectant des problèmes, et modifiant ses trajectoires au besoin.

Systèmes Experts Les programmes de diagnostic médical sont aujourd'hui capables de réaliser des diagnostics tout aussi fiables que les experts humains dans plusieurs spécialités médicales. On a même vu un expert humain rejeter le diagnostic proposé par un système expert sur un cas difficile, avant de reconnaître son erreur après que le programme eût expliqué son jugement.
Robotique Aujourd'hui, des robots sont régulièrement utilisés pour réaliser plusieurs types d'interventions chirurgicales, aidant les médecins à effectuer des manipulations plus précises et ouvrant la possibilité de faire de la chirurgie à la distance.

Chapitre 2 Algorithmes de Recherche

2.1 Introduction

La résolution des problèmes par des par des algorithmes de recherche définit un cadre formel pour de nombreux problèmes de l'industrie, de la finance ou de la vie quotidienne. Les problèmes sont habituellement définis comme une problématique de recherche d'une meilleure solution dans un ensemble très grand mais fini de solutions. Dans ce chapitre, on abordera les algorithmes de recherches. Après la présentation des principales définitions concernant la représentation des problèmes dans l'espace des états on y verra les deux grandes classes des algorithmes de recherche: les algorithmes non-informes (aveugles), les algorithmes informes.

2.2 Définition formelle d'un problème

Dans le contexte de ce chapitre, un problème sera défini par les cinq éléments suivants :

- un état initial
- un ensemble d'actions
- une fonction de successeur, qui définit l'état résultant de l'exécution d'une action dans un état un ensemble d'états buts
- une fonction de coût, associant à chaque action un nombre non-négative (le coût de l'action)

Nous pouvons voir un problème comme un graphe orienté où les nœuds sont des états accessibles depuis l'état initial et où les arcs sont des actions. Nous appellerons ce graphe l'espace des états. Une solution sera un chemin de l'état initial à un état but. On dit qu'une solution est optimale si la somme des coûts des actions du chemin est minimale parmi toutes les solutions du problème.

Nous passons tout de suite à des exemples concrets qui devraient vous permettre de mieux comprendre cette définition.

Exemple 1 (Le taquin). Pour ceux que le connaissent pas, le taquin est une sorte de puzzle. Nous commençons avec une grille 3X3 de neuf cases où sont placées huit tuiles étiquetées par les nombres 1 à 8, une des cases restant vide. Une tuile située à côté de la case vide peut être déplacée vers cette case. L'objectif du jeu est d'arriver à obtenir une certaine configuration des tuiles dans la grille. Voici une formalisation de ce problème :

- **Etats:** Des états sont des configurations des huit tuiles dans les neuf cases de la grille. Voir fig. 1 pour deux exemples.
- **Etat initial:** N'importe quel état pourrait être choisi comme l'état initial.
- **Actions:** Il y aura 4 actions possibles correspondant aux quatre façons de changer la position du carré vide : haut, bas, gauche, droite (remarquez que dans certaines configurations, il n'y aura que 2 ou 3 actions possibles).
- **Fonction de successeur:** Cette fonction spécifie les états résultants des différentes actions. Par exemple, la fonction va nous dire que l'exécution de l'action droite dans le premier état de fig. 1 produira le deuxième état de fig. 1.
- **Test de but** L'état but est unique et fixé au début du jeu (n'importe quel état peut être choisi comme état but, même si en pratique il s'agit de remettre les nombres dans l'ordre).

- **Coût des actions** Chaque déplacement d'une tuile a coût de 1 (pour trouver une solution avec le moins de déplacements)

Le taquin est souvent utilisé pour tester les algorithmes de recherche. En augmentant la taille de la grille, nous pouvons créer les problèmes de plus en plus complexes. Les algorithmes d'aujourd'hui arrivent à résoudre les taquins 3X3 et 4 X 4 (qui ont des espaces d'états de taille 181440 et d'environ 1, 3 milliard respectivement), mais les instances du taquin 5 X 5 restent difficiles.

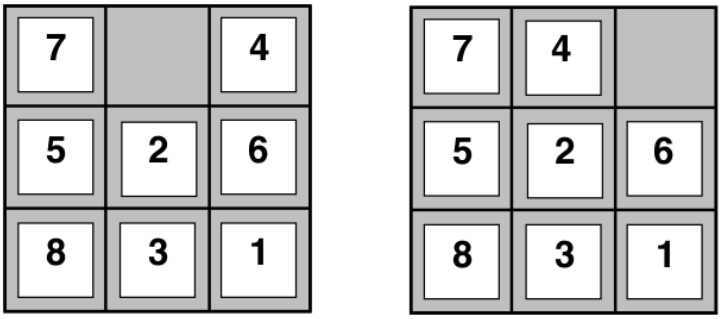


Fig. 1 – Deux états du jeu de taquin.

Exemple 2 (Huit reines). L'objectif de ce jeu est de placer huit reines sur un échiquier (une grille 8X8) tel qu' aucune reine attaque une autre reine, c'est à dire qu'il n'y a pas deux reines sur la même colonne, la même ligne, ou sur la même diagonale. La configuration donnée dans fig. 2 n'est donc pas une solution parce qu'il y a deux reines sur la même diagonale. Voici une première formalisation:

- **Etats:** Toute configuration de 0 à 8 reines sur la grille.
- **Etat initial:** La grille vide.
- **Actions:** Ajouter une reine sur n'importe quelle case vide de la grille.
- **Fonction de successeur:** La configuration qui résulte de l'ajout d'une reine à une case spécifié à la configuration courante.
- **Test de but:** Une configuration de huit reines avec aucune reine sous at-
- taque.
- **Coûts des actions:** Ce pourrait être 0, ou un coût constant pour chaque action, nous nous intéressons pas du chemin, seulement l'état but obtenu.

On peut réduire l'espace d'états drastiquement en observant qu'il est inutile de continuer de développer des configurations où il y a déjà à un conflit (comme on ne peut pas résoudre le conflit en ajoutant des reines supplémentaires). Voici les changements de notre formalisation:

- **Etat initials**:Configurations de n reines ($0 \leq n \leq 8$), une reine par colonne dans les n colonnes plus à gauche, avec aucun conflit.
- **Actions** Ajouter une reine à une case vide dans la colonne vide la plus à gauche dans une façon à éviter un conflit.
- Ces deux formalisations sont incrémentales car nous plaçons des reines une par une sur la grille. Mais ce n'est pas la seule façon de le faire. Une autre possibilité serait de commencer avec les huites reines sur la grille (une par colonne) et puis de changer la position d'une reine à chaque tour.
- **Etat initials:** Une configuration de 8 reines telle qu'il n'y a qu'une seule reine par colonne
- **Etat initial:** Un état choisi aléatoirement.
- **Actions** Changer la position d'une reine dans sa colonne.

Notons qu'avec les caractérisations incrémentales nous ne tombons jamais sur un état déjà visité, mais ce n'est pas le cas avec cette dernière formalisation (car nous pouvons changer la position d'une reine et après la remettre dans sa position d'origine) ni avec le taquin (où nous pouvons très bien déplacer une tuile et la remettre à sa place au coup suivant). Dans ces derniers cas, il nous faut faire attention pour ne pas explorer une deuxième fois un chemin déjà exploré ou de tourner en boucle.

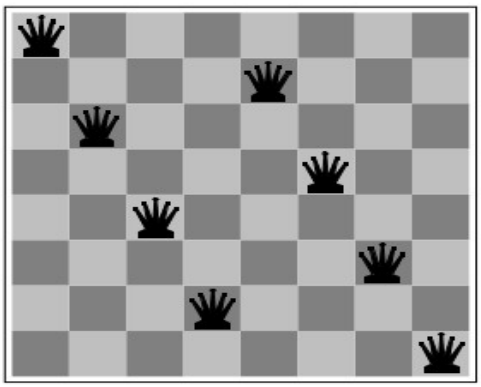
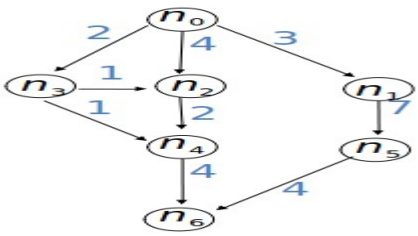


Fig. 2 – Une presque solution pour le problèmes des huit reines.

Exemple 3. Trouver le plis court chemin dans une ville.

n_0 : ville de départ (état initial) n_6 : destination (but) En d'autres termes : *goal*(n) : vrai si n=n_6 Routes entre les villes : *transitions*(n_0) = (n_3, n_2, n_1) Distance entre les villes : $c(n_0,n_2) = 4$	

2.3 Structure générale d'un algorithme de recherche

La plupart des algorithmes de recherche suivent à peu près le même schéma : nous commençons toujours dans l'état initial et puis nous exécutons
les étapes suivantes en boucle jusqu'à terminaison :
– s'il n'y a plus d'états à traiter, renvoyez échec
– sinon, choisir un des états à traiter (**)

- si l'état est un état but, renvoyez la solution correspondante
- sinon, supprimer cet état de l'ensemble des états à traiter, et le remplacer par ses états successeurs

2.4 Evalution des algorithmes de recherche

Dans la suite, nous allons différents d'algorithmes de recherche. Comment pouvons-nous les comparer ? Voici quatre critères que nous allons utiliser pour comparer les différents algorithmes de recherche :
Complexité en temps Combien du temps prend l'algorithme pour trouver la solution ?
Complexité en espace Combien de mémoire est utilisée lors de la recherche
d'une solution ?
Complétude Est-ce que l'algorithme trouve toujours une solution s'il y en a une ?
Optimalité Est-ce que l'algorithme renvoie toujours des solutions optimales ?

2.5 Un algorithme de recherche générique

Avant de considérer des algorithmes spécifiques, nous présentons un algorithme de recherche générique, dont le pseudo-code est donné dans Figure 1.

```
fonction RECHERCHE(état_initial, SUCCESSEURS, TEST_BUT, COÛT)
    noeuds_à_traiter ← CRÉER_LISTE(CRÉER_NOEUD(état_initial, [], 0))
    boucle
        si VIDE ?(noeuds_à_traiter) alors renvoyer échec
        noeud ← ENLEVER_PREMIER_NOEUD(noeuds_à_traiter)
        si TEST_BUT(ÉTAT(noeud)) = vrai
            alors renvoyer CHEMIN(noeud), ÉTAT(noeud)
        pour tout (action, état) dans SUCCESSEURS(ÉTAT(noeud))
            chemin ← [action, CHEMIN(noeud)]
            coût_du_chemin ← COÛT_DU_CHEMIN(noeud) + COÛT( état)
            s ← créer_noeud(état, chemin, coût_du_chemin)
            INSÉRER(s, noeuds_à_traiter)
```

Figure 3. Algorithme de recherche générique

L'algorithme prend en entrée la description d'un problème : un état initial, une fonction de successeurs, un test de but, et une fonction de coût. La première étape de l'algorithme est d'initialiser une liste 1 de nœuds à traiter, un nœud étant composé d'un état, d'un chemin, et du coût du chemin (voir Figure 2). Nous commençons avec un seul nœud correspondant à l'état initial.
A chaque itération de la boucle, nous vérifions si la liste de nœuds à traiter est vide. Si c'est le cas, nous avons examiné tous les chemins possibles sans pour autant trouver une solution, donc l'algorithme renvoie "échec". Si la liste contient encore des nœuds, nous sortons le premier nœud de la liste.

Si l'état de ce nœud est un état but, c'est gagné, et nous renvoyons l'état et le chemin qui permet d'accéder à ce nœud but. Dans le cas contraire, la recherche se poursuit : nous produisons les successeurs du nœud et les insérons dans la liste de nœuds à traiter.

2.6 Recherche non-informée

Dans cette section, nous introduisons des algorithmes de recherche non-informés. Ces algorithmes sont ainsi nommés parce qu'ils ne disposent pas d'informations supplémentaires pour pouvoir distinguer des états prometteurs (ce n'est pas le cas par exemple des programmes joueurs d'échecs qui ne peuvent explorer toutes les possibilités, et se concentrent donc à chaque étape sur un petit nombre de coups qui leur semblent être les meilleurs). En l'absence de telles informations, ces algorithmes font une recherche exhaustive de tous les chemins possibles depuis l'état initial.

2.6.1 Parcours en largeur

Le parcours en largeur est un algorithme de recherche très simple : nous examinons d'abord l'état initial, puis ses successeurs, puis les successeurs des successeurs, etc. Tous les nœuds d'une certaine profondeur sont examinés avant les nœuds de profondeur supérieure. Pour implémenter cet algorithme, il suffit de placer les nouveaux nœuds systématiquement à la fin de la liste de nœuds à traiter. Le fonctionnement du parcours en largeur sur un exemple est présenté sur la Figure 5.

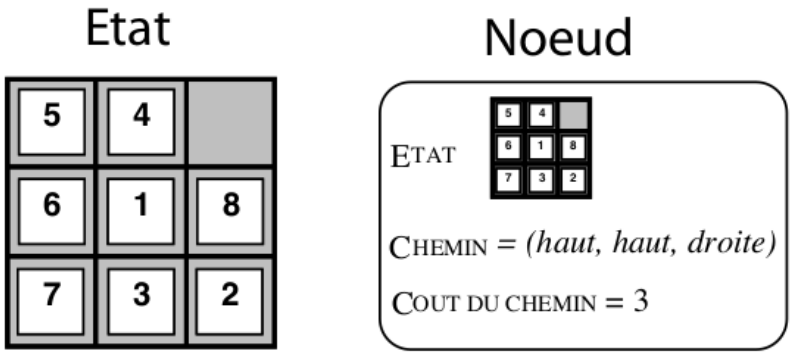


Figure 4 - Un nœud est composé d'un état, un chemin (une suite d'actions qui relie l'état à l'état initial), et le coût du chemin.

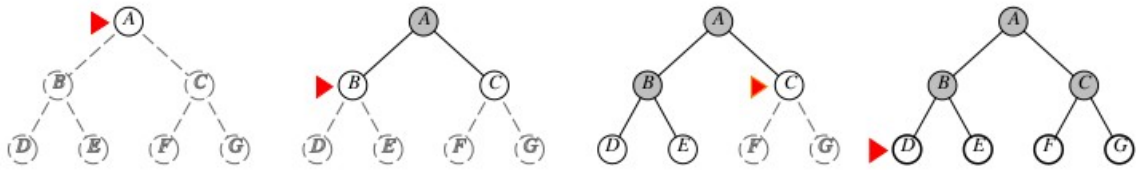


Figure 5 – Le parcours en largeur pour un arbre binaire simple. Nous com-mençons par l'état initial A, puis nous examinons les noeuds B et C de profondeur 1, puis les nœuds D, E, F, et G de profondeur 2.

Le parcours en largeur est un algorithme de recherche complet 2 à condition que le nombre de successeurs des états soit toujours fini (ce qui est très souvent le cas dans les problèmes courants). Pour voir pourquoi, soit p le nombre minimal d'actions nécessaires pour atteindre un état but depuis l'état initial. Comme nous examinons les nœuds profondeur par profondeur, et qu'il n'y a qu'un nombre fini de nœuds à chaque profondeur (ce ne serait pas le cas si un nœud pouvait avoir un infini de successeurs), nous atteindrons forcément le niveau p.

Le parcours en largeur trouve toujours un état but de plus petite profondeur possible. Donc si nous cherchons une solution quelconque, ou une solution avec le moins d'actions possibles, le parcours en largeur est optimal. Un peu plus spécifiquement, le parcours en largeur est optimal quand le coût du chemin ne dépend que du nombre d'actions de ce chemin. Par contre le parcours en largeur n'est pas optimal dans le cas général .

2.6.2 Parcours en profondeur

Le parcours en profondeur suit le chemin courant le plus longtemps possible. Il est facile à implémenter : il faut tout simplement mettre des successeurs du nœud courant au début de la liste de nœuds à traiter. La Figure 6 montre le fonctionnement de cet algorithme sur un petit exemple. Le

parcours en profondeur n'est pas complet parce que l'algorithme peut continuer sur un chemin infini, ignorant complètement un état but qui se trouve sur un autre chemin. Si par contre, nous n'avons qu'un nombre fini de chemins possibles (ce qui n'est pas souvent le cas), le parcours en profondeur sera complet. L'algorithme n'est pas optimal : il n'y a rien qui garantie que le premier état but trouvé sera le bon

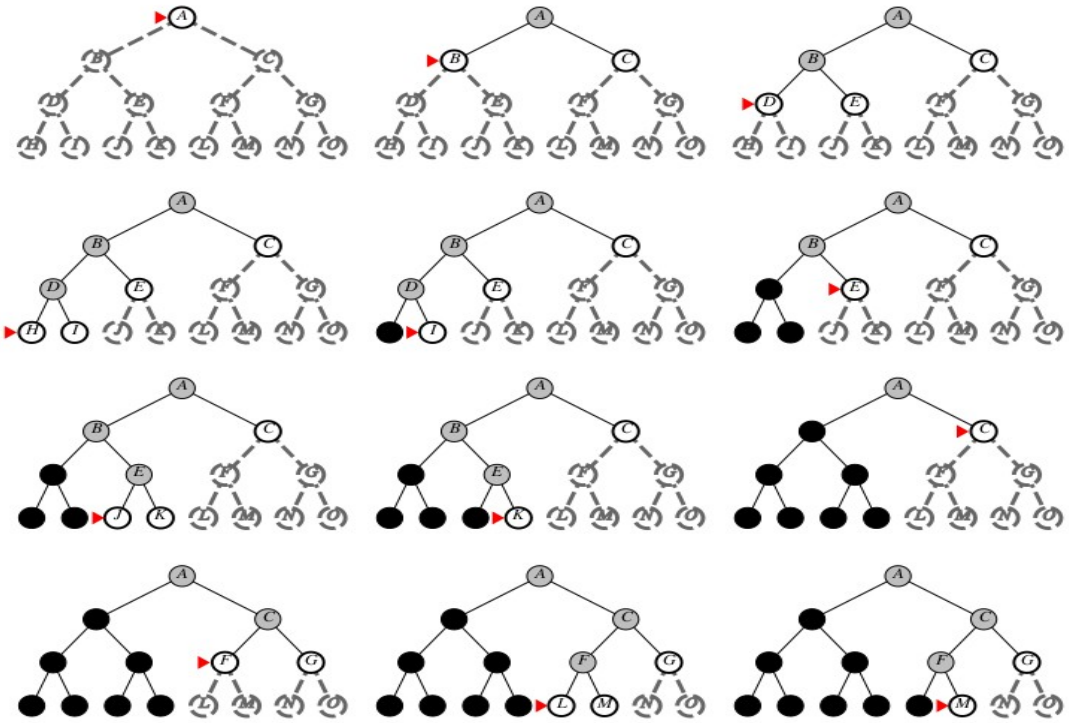


Figure 6 – Le parcoure en profondeur pour un arbre binaire simple. En gris le chemin en cours d'examen, en blanc les nœuds non-explorés, et en noir les nœuds explorés et rejetés.

2.7 Recherche Heuristique

Les algorithmes que nous avons vus dans la dernière section font une recherche exhaustive de tous les chemins possibles, ce qui les rend inefficaces voire inutilisables sur les problèmes de grande taille. Dans cette section, nous présentons les algorithmes de recherche heuristiques qui utilisent des informations supplémentaires pour pouvoir mieux guider la recherche.

Tout algorithme de recherche heuristique dispose d'une fonction d'évaluation f qui détermine l'ordre dans lequel les nœuds sont traités : la liste de nœuds à traiter est organisée en fonction des **f-valeurs** des nœuds, avec les nœuds de plus petite valeur en tête de liste. A priori, il n'y a pas vraiment de restriction sur la nature de la fonction d'évaluation, mais souvent elle a comme composante une fonction heuristique h où

h(n) = coût estimé du chemin de moindre coût reliant n à un état but

Notons que la fonction heuristique prend un nœud en entrée mais sa valeur ne dépend que de l'état associé au nœud. Et bien sûr, h(n) = 0 si n est un état but.

Prenons par exemple le problème suivant: nous sommes à Arad en Roumanie et il nous faut atteindre (en parcourant la plus petite distance possible) Bucarest. La Figure 7 présente les principales routes de la Roumanie et les distances associées. Soit d(A, B) la distance routière entre la ville A et la ville B, c'est-à-dire le plus court chemin entre la ville A et la ville B. On a par exemple d(Zerind, Sibiu) = 215. Pour mesurer à quel point nous sommes proches du but lorsque nous sommes dans la ville X, il serait bon de connaître d(X, Bucarest), et on aimerait alors prendre

h(X) = d(X, Bucarest). Le problème est que l'on n'a pas accès à la fonction d, et que son calcul n'est pas trivial (l'exemple ici étant de petite taille, calculer d serait faisable, mais sur un graphe de grande taille cela est beaucoup plus difficile). C'est ici qu'intervient l'heuristique, c'est-à-dire l'approximation : on approxime d(X, Bucarest) par h(X)=distance à vol d'oiseau entre la ville X et Bucarest (qui elle est très facilement calculable à l'aide d'une carte routière et d'un décimètre). Les valeurs de cette heuristique sont fournies dans la Figure 8. Comme les distances par la route sont toujours supérieures à la distance à vol d'oiseau, h(X) n'est qu'une approximation du coût réel. Si notre heuris- tique était parfaite, nous n'aurions même pas besoin de faire une recherche !

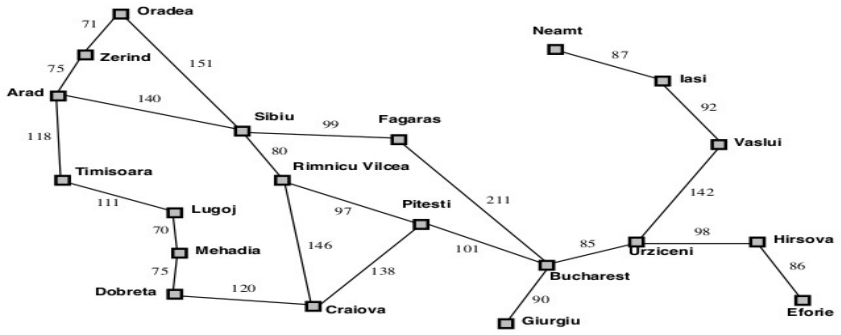


Figure 7. Une carte de route simplifié de la Roumanie.

Arad	366	Mehadia	241
Bucarest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesi	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 8. Des valeurs heuristiques : Distances à vol d'oiseau de Bucarest.

2.8 Recherche A*

Si nous appelons g(n) le coût du chemin entre l'état initial et n, la fonction d'évaluation utilisée par la recherche A* est donnée par la formule suivante:

f(n) = g(n) + h(n).

Comme g(n) est le coût réel associé au chemin entre l'état initial et n et que h(n) est une estimation du coût du chemin entre n et un état but, la fonction d'évaluation f donne une estimation du coût de la meilleure solution passant par le nœud n.

Figure 9 montre quelques étapes dans le déroulement de la recherche A* sur notre problème de trouver un chemin de Arad à Bucarest.

- Nous commençons dans l'état initial Arad. Comme Arad n'est pas un état but, nous considérons ses trois successeurs et comparons leurs valeurs de f : Sibiu (393=140 + 253), Timisoara (447 = 118 + 329), et Zerind (449 = 75 + 374).
- Nous choisissons Sibiu comme ce nœud a la plus petite valeur d'évaluation parmi tous les nœuds restants à ¸traiter. Nous ajoutons les quatre successeurs de Sibiu à la liste de noeud à traiter : Arad (646= 280 + 366), Fagaras (415=239 + 176), Oradea (671 = 291 + 380), et Rimnicu Vilcea (413= 220 +193).
- Le nœud le plus prometteur est Rimnicu Vilcea, donc nous ajoutons ses trois successeurs : Craiova (526= 366 + 163), Pitesti (417= 317 + 100), et Sibiu (553= 300 + 253) 4 . Nous comparons les différents nœuds à traiter, et
- nous choisissons Fagaras comme il a la plus petite valeur d'évaluation. Nous ajoutons les deux successeurs de Faragas à la liste de nœuds à traiter : Sibiu(591= 338 + 253) et Bucarest (450= 450 + 0).
- Remarquons que Bucarest est dans la liste de nœuds à traiter, mais nous n'arrêtons pas encore parce
- que nous avons toujours un espoir de trouver une solution de moindre coût. Nous continuons alors avec Pitesti qui a une valeur d'évaluation de 417 (qui est inférieur au coût de chemin à Bucarest que nous venons de trouver).
- Les trois successeurs de Pitesti sont ajoutés à la liste de noeuds à examiner : Bucarest (418=418 + 0), Craiova (615= 455 + 160), et Rimnicu Vilcea (607=414 + 193). Maintenant c'est le nouveau nœud Bucarest qui a la meilleure valeur d'évaluation, et comme Bucarest est un état but, l'algorithme termine et renvoie le chemin (Arad, Rimnicu Vilcea, Pitesti, Bucarest) associé à ce nœud.

L'algorithme de recherche A* est complet et optimal s'il y a un nombre fini de successeurs (on commence à avoir l'habitude....) et si nous plaçons une certaine restriction sur la fonction heuristique h. Il faut que la fonction h soit admissible, c'est à dire que la valeur h(n) ne doit jamais être supérieure au coût réel du meilleur chemin entre n et un état but. Notre heuristique prenant les distances à vol d'oiseau est un exemple d'une fonction heuristique admissible parce que la distance à vol d'oiseau (la valeur heuristique) n'est jamais supérieure à la distance par la route (le vrai coût).
Si nous voulons utiliser la technique proposée pour éliminer les nœuds.

Si nous voulons utiliser la technique proposée pour éliminer les nœuds redondants (selon laquelle nous ne gardons que le premier chemin ammènant à un état), il nous faut placer une restriction plus forte sur la fonction heu- ristique h pour garantir l'optimalité : la fonction doit être consistante. Une fonction heuristique est consistante si pour tout noeud n et tout successeur n' obtenu à partir n en faisant l'action a, nous avons l'inégalité suivante :

$$h(n) \leq c(a) + h(n')$$

Les fonctions heuristiques consistantes sont toujours admissibles, mais les fonctions heuristiques admissibles sont très souvent, mais pas toujours, consistantes.
La complexité de la recherche A* dépend de la fonction heuristique en question. En général, la complexité en temps et en espace est grande, ce qui rend la recherche A* mal adapté pour les problèmes de grande taille.

Pour pallier cet inconvénient, plusieurs autres algorithmes heuristiques moins gourmands en mémoire ont été proposés, mais nous ne les examinerons pas dans ce cours.

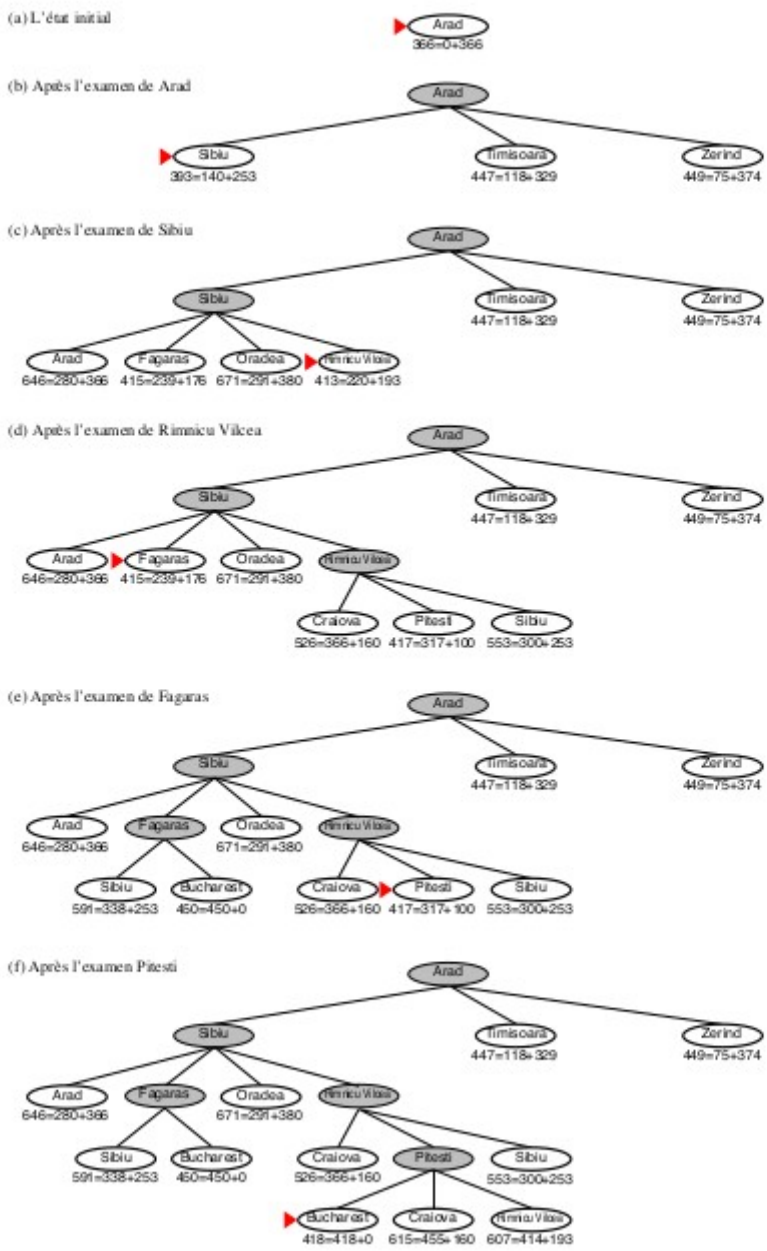


Figure 9 – La recherche A* pour trouver un chemin de Arad à Bucarest.

Chapitre 2: La logique propositionnelle et la logique des prédicats

Chapitre 4 Système Expert

4.1 Définition d'un système expert

Définition 4.1 (Expert). Un expert est une personne qui possède des compétences spécialisées, l'expérience et les connaissances que la plupart des gens n'ayant pas la capacité d'exploiter ces connaissances en utilisant astuces, raccourcis, et des règles de base pour résoudre un problème de façon efficace.

Définition 4.2 (Un système expert). Les systèmes experts représentent une application pratique de l'intelligence artificielle (AI). C'est un programme informatique qui est conçu pour supporter l'accumulation des connaissances d'un ou de plusieurs experts du domaine et capable de représenter le savoir par des règles de production.

4.2 Applications

Parmis les exemples de système expert, on cite

- **PUFF:** Système médical pour le diagnostic des affections respiratoires
- **PROSPECTOR:** un système utilisé par les géologues pour identifier des sites de forage ou d'exploitation minière.
- **MYCIN:** Système de soins de santé pour le diagnostic des troubles sanguins, il est utilisé pour la première fois en1979.
- **DESIGN ADVISOR:** Donne des conseils aux concepteurs de puces et de processeur.
- **DENDRAL:** utilisé pour identifier la structure des composés chimiques utilisé pour la première fois en 1965.
- **LITHIAN:** Donne des conseils aux archéologues pour l'examen des outils de pierre

4.3 Les composants de système expert

Le but d'un système expert est la modélisation d'un expert humain, faire des tâches d'analyse, de synthèse, et de résolution. Le cœur d'un système en I.A. est composé de trois parties:

1. **Une base de connaissances** où sont stockées, sous une forme appropriée, **toutes les connaissances** permettant de résoudre le problème que l'on veut traiter **dans un domaine déterminé**.

2. **Un mécanisme d'exploitation (moteur)** qui est un ensemble de programmes susceptibles de **traduire le raisonnement humain** (heuristique, progression par avancement et retour arrière, décomposition en sous ensemble...). Ces programmes utilisent les connaissances stockées dans la base de connaissance afin de résoudre le problème que l'utilisateur lui a posé en lui donnant des hypothèses de départ.

3. **Une base des faits** du problème à résoudre, qui est en fait une mémoire de travail où seront stockées les **hypothèses de départ** où viendront s'ajouter **des faits nouveaux** au fur et à mesure que progressera le mécanisme d'exploitation.

Un système, en I.A., doit posséder deux autres parties qui sont en fait des modules d'interface avec l'extérieur :

4. Un module d'interaction avec les utilisateurs potentiels: cette interface de dialogue doit être la plus conviviale possible. Ce module est lié au mécanisme d'exploitation.
5. **Un module d'aide à l'acquisition des connaissances:** qui est surtout une interface de dialogue entre la base de connaissances et les experts du domaine qui doivent alimenter cette dernière.

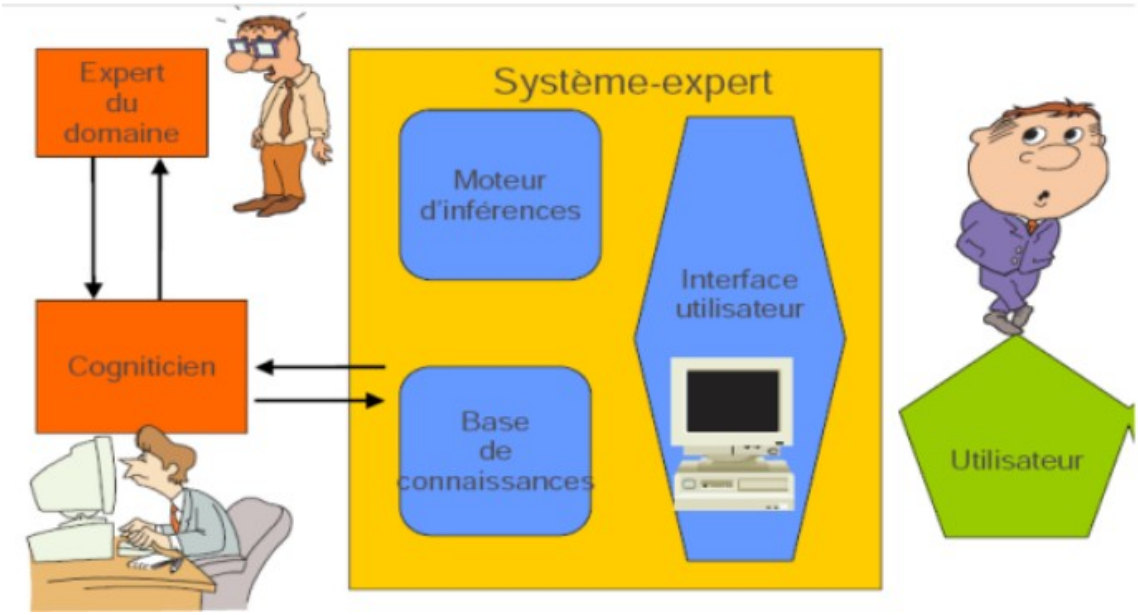


Figure 10: Architecture logicielle d'un Système expert

4.3.1 Base de Connaissances

Une description des objets constituant le contexte du domaine du système en I.A., et ceci sous plusieurs formes :

- Description-classification :
  "le canari est un oiseau".
- Description-structure :
  "le part-brise est un élément de la carrosserie d'une voiture, d'un camion, d'un car".
- Description-définition :
  "une chaise est un meuble qui a 4 pieds, 1 dossier et 1 siège".

Des connaissances opérationnelles sur les objets décrits précédemment ; elles indiquent :

- Des lois, des théorèmes :
  "la vitese est donnée par la distance parcourue sur le temps mis pour la parcourir".

- Des règles d'expertise :
  "si un oiseau est jaune, alors c'est un canari".
- Des stratégies :
  "si golf alors consulter les sports individuels".

La représentation des connaissances en I.A. :

- **Sous forme de règles de production (ou logique des propositions) :**
  SI...**ALORS**...
  si Jean est un homme **alors** Jean est mortel.
- **En logique du premier ordre ou le calcul des prédicats :**
  Homme(x) **alors** mortel(x).

La représentation des connaissances est liée au sujet traité.

- **La logique propositionnelle ou du premier ordre** pour la démonstration automatique de théorèmes mathématiques.
- **Les réseaux sémantiques** pour l'analyse du langage naturel, pour la traduction automatique.
- **Les règles de production** pour la description d'un domaine circonscrit parfaitement connu par un expert.
- **etc...**

4.3.2 Base de faits:

Ce sont en quelque sorte les formules (les faits) dont le contenu (valeur de vérité) est connu par le système (ou l'utilisateur). On peut dire que les faits consistent à attribuer une interprétation qui correspond à une in- stance du domaine considéré. C'est un espace de travail, modifié par les nouveaux faits déduits par le raisonnement. Toute formule déduite ou prouvée est considérée comme un fait. Les faits peuvent prendre des formes plus au moins complexes.Nous envisagerons que des faits élémentaires dont les valeurs possibles sont:

- **Booléennes** : vrai, faux
- **Symboliques**:c'est -à-dire appartenant à un domaine fini de symboles
- **Réelles** : pour représenter les faits continus

Par exemple, actif est un fait booléen, profession est un fait symbolique et rénumération est un fait réel.

Exemple 4.1

système expert SPHINX.
FAITS : (fièvre 40°, boutons, amaigrissement).

Exemple 4.2

système expert EXPERT.
FAITS : (ANIMAL a des plumes),
(ANIMAL vole)

Un système expert qui n'utilise que des faits booléens est dit d'ordre 0. Un système expert utilisant les variables est dit d'ordre 1. Pour qu'un système expert puisse modéliser un raisonnement humain,

il est in dispensable qu'il puisse raisonner sur ses propres raisonnements, réfléchir aux faits qu'il manipule, aux formules qu'il peut construire, etc.

Autrement dit, La base des faits s'enrichira progressivement à la demande du système expert pour qu'il puisse progresser vers la solution. Par exemple, avec les faits initiaux de l'exemple 2, le système expert trouve le fait que ANIMAL est oiseau. La base des faits contiendra : (ANIMAL a des plumes), (ANIMAL vole), (ANIMAL est oiseau).

Cette base des faits peut jouer le rôle de mémoire auxiliaire, en mémorisant tous les résultats intermédiaires. Ils permettent de conserver une trace des raisonnements effectués par le système. Ils peuvent expliquer l'origine des connaissances déduites par le système expert.

4.3.3 Base de règles

Elle rassemble la connaissance et le savoir-faire de l'expert, par conséquent elle ne change pas au cours d'une session de travail. La forme de la règle est comme suit :

Si conjonction de conditions alors Action

Condition : (encore appelées prémisses), teste l'appartenance des faits à la base des faits.

Action : un effet sur la base de faits (ajout ou suppression d'un fait). Ne modifie pas la base des règles.

Exemple 1.3

SI l'animal pond des oeufs et l'animal vole **ALORS** l'animal est un oiseau.

La partie gauche est une prémisse. La partie droite est une conclusion. Une telle règle est aussi appelée inférence. La partie gauche exprime des conditions. Si toutes les conditions sont satisfaites l'autre partie devient vraie.

4.3.4 Comment fonctionne en pratique un système expert ?

Exemple 1.4 : **Système expert ANIMAUX**

Base des règles "ANIMAUX" :	Base des faits "ANIMAUX" :
● **R1 : SI** l'animal est oiseau et l'animal pèse moins de 10 gr **ALORS** l'animal est colibri. ● **R2 : SI** l'animal vole et l'animal pond des œufs **ALORS** l'animal est oiseau	● Animal vole ● Poids animal est inférieur à 10 gr ● Animal pond des oeufs

Le mécanisme d'exploitation ou moteur d'inférence prend une règle et regarde dans la base de faits si les faits stockés vont rendre la prémisse de la règle vraie.

Tirons la première règle **R1** : l"une des conditions n'est pas remplie. En effet, on ne sait pas si l'animal est oiseau. Ce fait n'est pas dans la base des faits.
Tirons la règle suivante **R2** : les conditions sont vraies. La base des faits contient les deux conditions animal vole et animal pond des œufs. La règle est exécutée.
La base des faits s'enrichit d'un fait supplémentaire : "l'animal est oiseau".

Tirons maintenant de nouveau la règle **R1** : les conditions sont vraies, la règle est éxécutée. Un fait nouveau est ajouté à la base des faits : "l'animal est colibri".

Exemple 1.5 : **Système expert VILLE**

la base de règles "ville" :	**la base des faits "ville":**
R1 : Si belle ville et très bon restaurants **alors** ville métitant le voyage. • **R2 : Si** ville historique **alors** ville méritant le voyage. • **R3 : Si** autochtones accueillants et traditions folkloriques **alors** ville méritant le voyage. • **R4 : Si** monuments et végétation abondante **alors** belle ville. • **R5 : Si** tradition culinaire **alors** bons restaurants. • **R6 : Si** restaurants 3 étoiles **alors** très bons restaurants. • **R7 : Si** restaurants 3 toques **alors** très bons restaurants. • **R8 : Si** musées et ville ancienne **alors** ville historique. • **R9 : Si** Provence et bord de mer **alors** autochtones accueillants. • **R10 : Si** parcs verdoyants et avenues larges **alors** végétation abondante.	Parcs verdoyants • Avenues larges • monuments • restaurants 3 toques • ville ancienne

4.3.4 Moteur d’inférences :

Un moteur d’inférence est un mécanisme qui permet d’inférer des connaissances nouvelles à partir de la base de connaissance du système. Le moteur d’inférence peut fonctionner selon deux modèles : le chaînage avant et le chaînage arrière.

Le chaînage avant Le principe du chaînage avant est simple, il requiert l’accès aux prémisses (standards d’engagement) afin de déclencher les règles d’inférence adéquates. L’application des

règles (évaluations) donnent des résultats, ceux-ci sont évalués afin de savoir si l'on a accédé à une solution finale potentielle.

- Si c'est le cas, la solution est proposée à l'utilisateur. S'il la valide, la solution est enregistrée dans la base de faits comme solution, sinon comme simple résultat et on continue dans le cas suivant.
- Si cela n'est pas le cas ou si la solution est refusée, la solution est enregistrée dans la base de faits comme simple résultat et le moteur d'inférence tente d'y appliquer d'autres règles jusqu'à trouver une solution potentielle validée, ou jusqu'à ce qu'il n'y ait plus de règle.

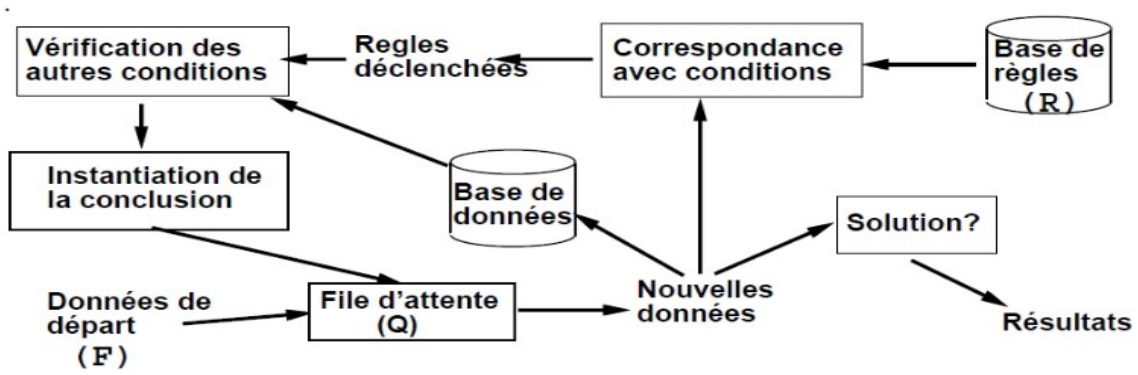


Figure 11: Composants d'un moteur d'inférence avant

Une règle est déclenchable quand la pré-condition est évaluée à vrai en accord avec l'interprétation donnée dans la base des faits.

Exemple 1.6 : **Système expert ABC**

Soit la base de connaissances suivante :

Base de faits initiale :

{H,K}

Base des règles :

- **R1 :** A --> E
- **R2 :** B --> D
- **R3 :** H --> A
- **R4 :** E et G --> C
- **R5 :** E et K --> B
- **R6 :** D et E et K --> C
- **R7 :** G et K et F --> A

on obtient la chaîne de dérivation suivante :

- H --> A (R3) et la base de faits B = {A,H,K}
- A --> E (R1) et B = {A,E,H,K}
- E et K --> B (R5) et B = {A,B,E,H,K}
- B --> D (R2) et B = {A,B,D,E,H,K}
- D et E et K --> C (R6) et B = {A,B,C,D,E,H,K}

Le chaînage arrière Le principe du chaînage arrière est plus compliqué, il s'agit dans ce cas de partir d'un effet ou d'une solution et de tenter de remonter la chaîne afin de déterminer les causes d'un effet (fait). La procédure est à partir d'un fait, de déterminer, grâce aux metarules, les règles d'inférence qui auraient pu être à l'origine de ce fait et de déterminer les paramètres les plus probables. A partir de là, on analyse les paramètres :

- Si le paramètre est un fait enregistré dans la base de faits, c'est qu'il est le résultat d'une règle (évaluation). La procédure précédemment décrite est donc relancée.

- Si le paramètre n'est pas un fait de la base de faits, on en reste là. On relève alors tous les faits et données retrouvés. Ils représentent les causes probables de la conséquence étudiée.

Dans l'exemple précédent :

- On considère la règle (R2) et on définit B comme nouveau but.
- On considère alors la règle (R5) et on définit E comme nouveau but (puisque K est dans la base des faits).
- On considère alors la règle (R1) et on définit E comme nouveau but.
- On considère alors les règles (R3) et (R7). (R3) permet de conclure.

Conclusion

Les systèmes experts sont une des applications de l'intelligence artificielle qui ont quitté les laboratoires de recherche pour être utilisées dans le monde professionnel. De nombreux systèmes experts ont été implantés avec succès pour résoudre des problèmes concrets.

Chapitre 5 Exercices
Exercice 1
Lequel des éléments suivants signifie la même chose que "p est nécessaire pour q"?
1. p⇦q
2.p⇒q
3.q⇒p

Lesquelles des conditions suivantes sont nécessaire pour « le nombre naturel n est un multiple de 10 »?

1. n est un multiple de 5
2. n*n est un multiple de 100
3. n est pair et un multiple de 5
4. n est un multiple de 20
5. n = 100

Lesquelles des conditions suivantes sont suffisantes pour « le nombre naturel n est un multiple de 10 »?

1. n est un multiple de 5
2. n*n est un multiple de 100
3. n est pair et un multiple de 5
4. n est un multiple de 20
5. n = 100

Exercice 2
Exercice 2
Représenter les connaissances suivantes en logique des prédicats ...
1a. Finn est chez-lui ou chez Rey.
1b. Si Finn n'est pas chez-lui, il est chez Rey.
2a. Vous pouvez déduire vos frais médicaux si votre revenu annuel est inférieur à 18 000€
2b. Vous ne pouvez pas déduire vos frais médicaux si vous n'avez pas plus de 70 ans ou que votre
3a. Jean réussira son examen ou il n'est pas fort en logique.
3b. Si Jean ne réussit pas son examen alors il n'est pas fort en logique.
3c. Si Jean n'est pas fort en logique, alors il ne réussit pas son examen.
4a. Si Jean n'est pas fort en logique, Marie n'est pas forte non plus en logique et ils ne réussiront pas
4b. Jean et Marie réussiront leur examen s'ils sont forts en logique.
5a. Chargeur branché, électricité consommée. Électricité consommée, chaleur dissipée. Chaleur dissipée
5b. Chargeur branché, tempête en Bretagne.

Exercice 3
Représenter les connaissances suivantes avec les connecteurs logiques :
1. p sinon q
2. p à moins que q

3. p autrement q
4. Il suffit que p pour q
5. Il est nécessaire que p pour q 6. p seulement si q
7. p si q

Exercice 4
Représentez à l'aide de la logique des prédicats les informations suivantes :
1. Chaque chien a mordu au moins un facteur.
2. Tous les étudiants sont venus au cours d'IA.
3. Tous les étudiants ont testé toutes les boîtes.

Exercice 65
Soient P et Q les affirmations suivantes :
P = Ahmed est fort en Maths;
Q = Ahmed est fort en Chimie
Représenter sous formes symboliques les affirmations suivantes (utiliser les connecteurs Λ V $\rightarrow$) :
1. Ahmed est fort en Maths mais faible en Chimie
2. Ahmed n'est ni fort en Maths ni en Chimie
3. Ahmed est fort en Maths s'il est fort en Chimi.

Exercice 6
Un expert a construit la base de règles suivantes :
− R1: A et B $\rightarrow$ C
− R2:D $\rightarrow$ A
− R3:E $\rightarrow$ F
− R4:G $\rightarrow$ H
− R5:I $\rightarrow$ F
− R6:H et F et J $\rightarrow$ B
− R7:H et K $\rightarrow$ J
− R8: G et F $\rightarrow$ K

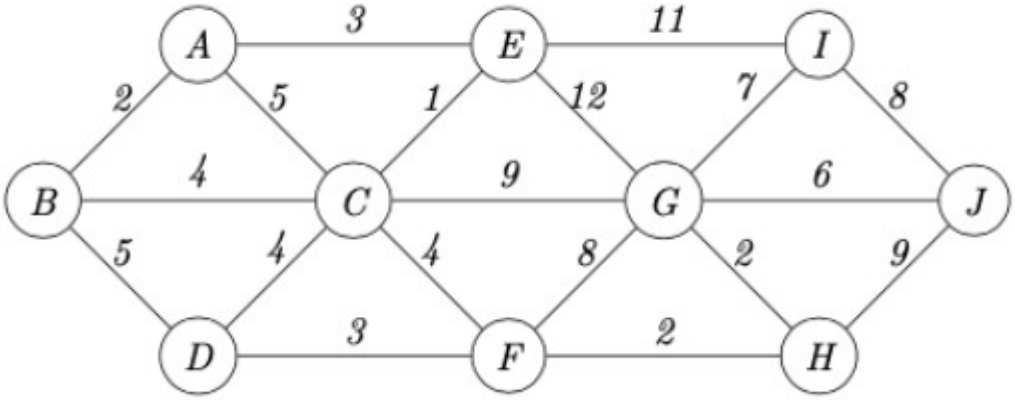
La base initiale de faits est : (D, G, I)

1. On veut prouver le fait C en chaînage avant ; quelle est la suite des faits prouvés en admettant que l'on parcourt la base de règles dans l'ordre dans laquelle elle est écrite et qu'un fait établi peut être utilisé immédiatement

2. On veut prouver le fait C par chaînage arrière; quelle est la suite de règles essayées pour prouver le fait C ? On indiquera si chaque règle essayée a été un succès ou un échec.

Exercice 7
Tracer l'exécution du chaînage avant sur l'exemple suivant :
F1: père(Jamal, Ahmed),
F2: frére(Ahmed, Fouad),
F3: frère(Jamal, Ali)
une base de connaissances qui consiste en deux règles:
R1: père(?x,?y) ∧ frère(?y,?z) ⇒ père(?x,?z)
R2: père(?x,?y) ∧ frère(?x,?z) ⇒ oncle(?z,?y) et le but de l'inférence:
oncle(?x, Fouad)

Exercice 8
Soit le graphe suivant, la valeur portée sur chaque arc correspond au coût de passage d'une extrémité de l'arc à l'autre. On souhaite calculer le plus court chemin de A à H

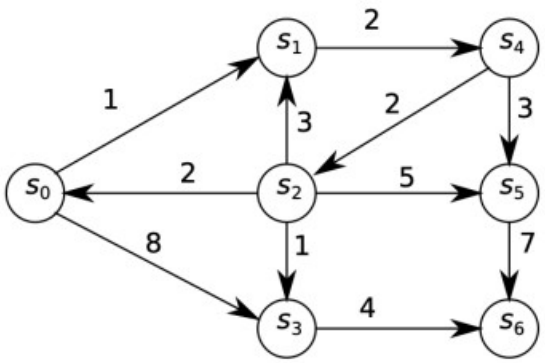


On a de plus la fonction heuristique h qui estime le coût pour atteindre H depuis chaque sommet. h est donnée par le tableau ci dessous.

A	B	C	D	E	F	G	H	I	J
9	7	3	2	6	1	2	0	4	6

1. Appliquez l'algorithme A* avec la fonction h sur ce graphe.
2. Donnez le plus court chemin de A à H ainsi que sa valeur que vous avez trouvés dans la question précédente.

Exercice 9
Soit l'espace d'états, les transitions, les fonctions heuristiques h1 et h2, et le but G suivants.



État	$h_1()$	$h_2()$
s_0	9	10
s_1	8	9
s_2	5	6
s_3	0	3
s_4	7	10
s_5	7	6
s_6	0	0

$G = \{s_6\}$

(a) Faites la trace de l'algorithme A* en utilisant l'état initial s0, l'heuristique h1 et le but G.
(b) L'heuristique h1 est-elle admissible ? Justifiez.
(c) Toujours en supposant le but G, l'heuristique h2 est-elle admissible ? Justifiez

Corrections :

Exercice 1
Lequel des éléments suivants signifie la même chose que "p est nécessaire pour q"?
1. p ⇦q
2.p ⇒q
3.q ⇒p

Lesquelles des conditions suivantes sont nécessaire pour « le nombre naturel n est un multiple de 10 »?

1. n est un multiple de 5
2. n*n est un multiple de 100
3. n est pair et un multiple de 5
4. n est un multiple de 20
5. n = 100

Lesquelles des conditions suivantes sont suffisantes pour « le nombre naturel n est un multiple de 10 »?

1. n est un multiple de 5
2. n*n est un multiple de 100
3. n est pair et un multiple de 5
4. n est un multiple de 20
5. n = 100

Exercice 2

- 1a. ahmed est chez-lui ou chez ali.
- 1b. Si ahmed n'est pas chez-lui, il est chez ali.
- 2a. Vous pouvez déduire vos frais médicaux si votre revenu annuel est inférieur à 18 000DA et que vous avez plus de 70 ans.
- 2b. Vous ne pouvez pas déduire vos frais médicaux si vous n'avez pas plus de 70 ans ou que votre revenu annuel est inférieur à 18 000€.
- 3a. mohammed réussira son examen ou il n'est pas fort en logique.
- 3b. Si mohammed ne réussit pas son examen alors il n'est pas fort en logique.
- 3c. Si mohammed n'est pas fort en logique, alors il ne réussit pas son examen.
- 4a. Si mohammed n'est pas fort en logique, Hesna n'est pas forte non plus en logique et ils ne réussiront pas leur examen.
- 4b. Mohammed et Hesna réussiront leur examen s'ils sont forts en logique.
- 5a. Chargeur branché, électricité consommée. Electricité consommée, chaleur dissipée. Chaleur dissipée, fonte de banquise. Fonte de banquise, tempête en Bretagne.
- 5b. Chargeur branché, tempête en Bretagne.

Solution

- 1a. setrouve(ahmed,maisonFrancois)∨setrouve(ahmed,maisonAli)
  1b. ¬setrouve(ahmed,maisonahmed)→setrouve(ahmed,maisonali)
- 2a. inferieur(revenu,18000)∧¬inferieur(age,70)→effectuer(deduction,fraisMedicaux
- 2b. inferieur(age,70)∨ inferieur(revenu,18000)→¬effectuer(deduction,fraisMedicaux)
- 3a. reussir(Mohammed,exam)∨¬fort(Mohammed,logique)
- 3b. ¬reussir(Mohammed,exam)→¬fort(Mohammed,logique)
- 3c. ¬fort(Mohammed,logique)→¬reussir(Mohammed,exam)
- 4a. ¬fort(Mohammed,logique)→¬fort(Hesna,logique)∧¬reussir(Mohammed,exam)∧¬reussi r(Hesna,exam)
- 4b. fort(Mohammed,logique)∧fort(Hesna,logique)→reussir(Mohammed,exam)∧reussr(Hes na,exam)
- 5a.
  est(chargeur,branché)→est(électricité,consommée)
  est(électricité,consommée)→est(chaleur,dissipée)
  est(chaleur,dissipée)→est(banquise,fondue)
  est(banquise,fondue)→est(tempete,bretagne)
- 5b. est(chargeur,branché)→est(tempete,bretagne)

Exercice 3

Représenter les connaissances suivantes avec les connecteurs logiques :

1. p sinon q
2. p à moins que q3.
3. p autrement q
4. Il suffit que p pour q
5. Il est nécessaire que p pour q 6. p seulement si q
6. p si q

Solution

1. **q → p**
2. **p → q**
3. **q → p**
4. **p → q**
5. **q → p**
6. **q <-> p**
7. **p → q**

Exercice 4

Représentez à l'aide de la logique des prédicats les informations suivantes :

1.Chaque chien a mordu au moins un facteur.

2.Tous les étudiants sont venus au cours d'IA.

3.Tous les étudiants ont testé toutes les boîtes.

Solution

- $\forall x, \exists y, est(x,chien) \wedge est(y,facteur) \rightarrow aMordu(x,y)$
- $\forall x, est(x,etudiant) \rightarrow aAssisté(x,coursIA)$
- $\forall x, \forall y, est(x,etudiant) \wedge est(y,boite) \rightarrow aTesté(x,y)$

Exercice 6

Soient P et Q les affirmations suivantes :

1. P = Ahmed est fort en Maths;
2. Q = Ahmed est fort en Chimie

Représenter sous formes symboliques les affirmations suivantes (utiliser les connecteurs Λ V $\rightarrow$) :

1. Ahmed est fort en Maths mais faible en Chimie
2. Ahmed n'est ni fort en Maths ni en Chimie
3. Ahmed est fort en Maths s'il est fort en Chimi.

Exercice 7

Un expert a construit la base de règles suivantes :

− R1: A et B $\rightarrow$ C
− R2:D $\rightarrow$ A
− R3:E $\rightarrow$ F
− R4:G $\rightarrow$ H
− R5:I $\rightarrow$ F
− R6:H et F et J $\rightarrow$ B
− R7:H et K $\rightarrow$ J
− R8: G et F $\rightarrow$ K

La base initiale de faits est : (D, G, I)

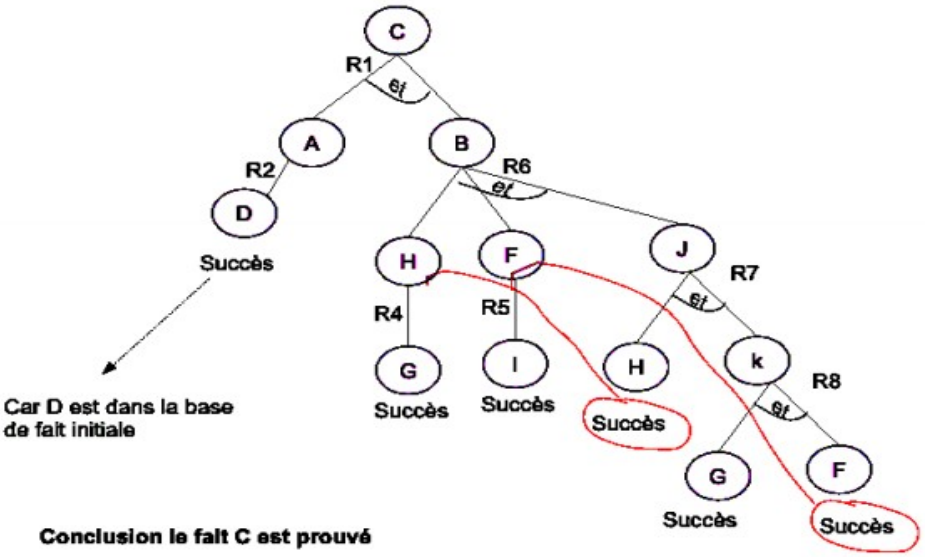
1. On veut prouver le fait C en chaînage avant ; quelle est la suite des faits prouvés en admettant que l'on parcourt la base de règles dans l'ordre dans laquelle elle est écrite et qu'un fait établi peut être utilisé immédiatement
2. On veut prouver le fait C par chaînage arrière; quelle est la suite de règles essayées pour prouver le fait C ? On indiquera si chaque règle essayée a été un succès ou un échec.

Solution

1.

Cycle	Base de faits	Règles déclenchées	Faits ajoutés
1	D, G, I	R2,R4,R5	A,H,F
2	D, G, I, A,H,F	R8	K
3	D, G, I, A,H,F, K	R7	J
4	D, G, I, A,H,F, K,J	R6	B
5	D, G, I, A,H,F, K,J,B	R1	C
6	D, G, I, A,H,F, K,J,B,C		

2.

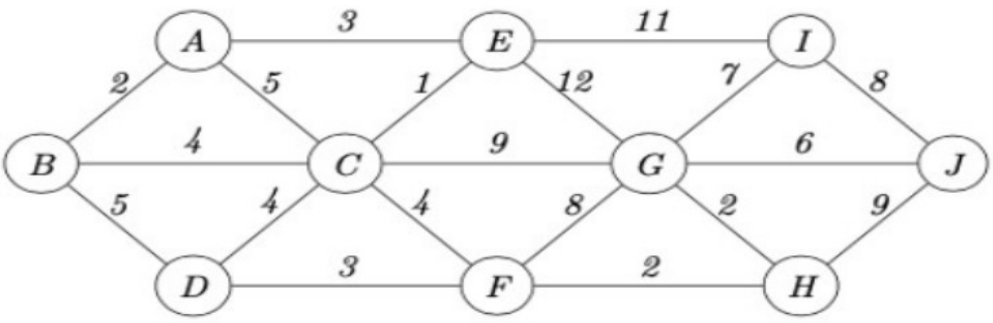


Exercice 8

- Tracer l'exécution du chaînage avant sur l'exemple suivant :
- F1: père(Jamal, Ahmed),
- F2: frére(Ahmed, Fouad),
- F3: frère(Jamal, Ali)
- une base de connaissances qui consiste en deux règles:
- R1: père(?x,?y) ∧ frère(?y,?z) ⇒ père(?x,?z) frère(?y,?z) ⇒q père(?x,?z)
- R2: père(?x,?y) ∧ frère(?y,?z) ⇒ père(?x,?z) frère(?x,?z) ⇒q oncle(?z,?y) et le but de l'inférence: oncle(?x, Fouad)

Exercice 9 :

Soit le graphe suivant, la valeur portée sur chaque arc correspond au coût de passage d'une extrémité de l'arc à l'autre. On souhaite calculer le plus court chemin de A à H



On a de plus la fonction heuristique h qui estime le coût pour atteindre H depuis chaque sommet. H est donnée par le tableau ci dessous

A	B	C	D	E	F	G	H	I	J
9	7	3	2	6	1	2	0	4	6

1. Appliquez l'algorithme A* avec la fonction h sur ce graphe.
2. Donnez le plus court chemin de A à H ainsi que sa valeur que vous avez trouvés dans la question précédente.

Solution

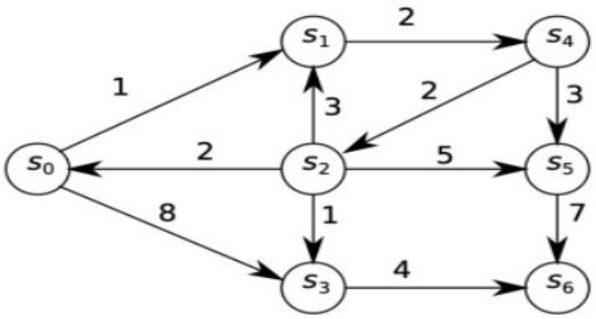
1. On représente le déroulement de l'algorithme sous forme de tableau. Chaque ligne du tableau correspond à un tour de boucle de l'algorithme. La première colonne indique le numéro de tour de l'algorithme, dans la deuxième colonne on indique le sommet choisi. Dans la troisième colonne l'ensemble des ouverts de l'étape courante est représenté, chaque état est noté sous le format @ sommet(g(x), f (x) = g(x) + h(x)) A. La dernière colonne contient l'ensemble des fermés de l'étape courante.

Etape	Choix	Ouverts	Fermés
Init		$\{A(0,9)\}$	$\emptyset$
1	$A(0,9)$	$\{B(2,9); C(5,8); E(3,9)\}$	$\{A(0,9)\}$
2	$C(5,8)$	$\{B(2,9); E(3,9); D(9,11); F(9,10); G(14,16)\}$	$\{A(0,9); C(5,8)\}$
3	$B(2,9)$	$\{E(3,9); \mathbf{D(7,9)}; F(9,10); G(14,16)\}$	$\{A(0,9); C(5,8); B(2,9)\}$
4	$E(3,9)$	$\{D(7,9); F(9,10); G(14,16); \mathbf{C(4,7)}; I(14,18)\}$	$\{A(0,9); B(2,9); E(3,9)\}$
5	$C(4,7)$	$\{D(7,9); \mathbf{F(8,9)}; \mathbf{G(13,15)}; I(14,18)\}$	$\{A(0,9); B(2,9); E(3,9); C(4,7)\}$
6	$D(7,9)$	$\{F(8,9); G(13,15); I(14,18)\}$	$\{A(0,9); B(2,9); E(3,9); C(4,7); D(7,9)\}$
7	$F(8,9)$	$\{G(13,15); I(14,18); H(10,10)\}$	$\{A(0,9); B(2,9); E(3,9); C(4,7); D(7,9); F(8,9)\}$
8	$H(10,10)$		

- A l'initialisation, on met dans Ouverts, le sommet de départ ;
- A chaque étape on choisit dans Ouverts un sommet s tel que f(s) = g(s) + h(s) soit minimal. Pour tous les voisins v de s, si v n'appartient ni à Ouverts ni à Fermés, on ajoute v à Ouverts. Sinon on remet v dans Ouverts avec une nouvelle valeur de g(v) seulement si g(s) +cout(s− > v) est inférieur à la valeur de g(v) mémorisée.
- A l'étape 3, après sélection du sommet B, la valeur de g(D) dans Ouverts passe de 9 à 7 ;
- A l'étape 4, après sélection du sommet E, la valeur de g(C) passe de 5 à 4 et C passe des Fermés aux Ouverts ;
- A l'étape 5, les valeur de g de F et G dans Ouverts passent respectivement de 9 à 8 et de 14 à

2. le plus court chemin de A à H est A pour un coût de 10.

Exercice 10
Soit l'espace d'états, les transitions, les fonctions heuristiques h1 et h2, et le but G suivants.



État	$h_1()$	$h_2()$
s_0	9	10
s_1	8	9
s_2	5	6
s_3	0	3
s_4	7	10
s_5	7	6
s_6	0	0

$G = \{s_6\}$

(a) Faites la trace de l'algorithme A* en utilisant l'état initial s0, l'heuristique h1 et le but G.
(b) L'heuristique h1 est-elle admissible ? Justifiez.
(c) Toujours en supposant le but G, l'heuristique h2 est-elle admissible ? Justifiez

Solution
(a) Trace d'exécution de A*

Étape	État choisi	Liste *open* $(s, f, g, parent)$ triée par f	Liste *closed* $(s, g, parent)$
#0	–	$(s_0, 9, 0, -)$	
#1	s_0	$(s_3, 8, 8, s_0), (s_1, 9, 1, s_0)$	$(s_0, 0, -)$
#2	s_3	$(s_1, 9, 1, s_0), (s_6, 12, 12, s_3)$	$(s_0, 0, -), (s_3, 8, s_0)$
#3	s_1	$(s_4, 10, 3, s_1), (s_6, 12, 12, s_3)$	$(s_0, 0, -), (s_1, 1, s_0), (s_3, 8, s_0)$
#4	s_4	$(s_2, 10, 5, s_4), (s_6, 12, 12, s_3), (s_5, 13, 6, s_4)$	$(s_0, 0, -), (s_1, 1, s_0), (s_3, 8, s_0), (s_4, 3, s_1)$
#5	s_2	$(s_3, 6, 6, s_2), (s_6, 12, 12, s_3), (s_5, 13, 6, s_4)$	$(s_0, 0, -), (s_1, 1, s_0), (s_2, 5, s_4), (s_4, 3, s_1)$
#6	s_3	$(s_6, 10, 10, s_3), (s_5, 13, 6, s_4)$	$(s_0, 0, -), (s_1, 1, s_0), (s_2, 5, s_4), (s_3, 6, s_2), (s_4, 3, s_1)$
#7	s_6	$(s_5, 13, 6, s_4)$ Solution trouvée : $\langle s_0, s_1, s_4, s_2, s_3, s_6 \rangle$ Coût : 10	$(s_0, 0, -), (s_1, 1, s_0), (s_2, 5, s_4), (s_3, 6, s_2), (s_4, 3, s_1), (s_6, 10, s_3)$

(b) Oui, l'heuristique h1 est admissible. Elle ne surestime jamais le coût restants. h∗(s0) = 10, h∗(s1) = 9, h∗(s2) = 5, h∗(s3) = 4, h∗(s4) = 7, h∗(s5) = 7, h∗(s6) = 0.

(c) Non, l'heuristique h2 n'est admissible, puisque la fonction heuristique h2 surestime le coût restant dans les états {s2,s4}.

Exercice 11

Construire le graphe du chaînage mixte appliqué à cette base de connaissances

Base de Règles

- R1 : SI Tropiques ALORS Les_Saintes
- R2 : SI Saint-Bart et hôtel ALORS Hôtel Paradisio
- R3 : SI dépressif ALORS Tourisme chaud
- R4 : SI tourisme chaud ALORS tropiques
- R5 : SI Les_Saintes ALORS Hôtel Paradisio
- R6 : SI Les_Saintes ALORS tourisme chaud
- R7 : SI P.D.G. ALORS tourisme chaud
- R8 : SI tourisme chaud et Les_Saintes ALORS tourisme chaud et voilier
- R9 : SI Hôtel Paradisio ALORS Caraïbes
- Base de faits : Les_Saintes

Exercice 12

Soit le programme Prolog suivant :

```
oiseau(pigeon).
oiseau(hirondelle).
carnivore(loup).
carnivore(lion).
animal(lion).
animal(X) :-oiseau(X).
manger(X,Y) :-carnivore(X),animal(Y), X\=Y.
```

1. Donner la solution de la requête : manger(lion,Y).
2. Donner l'arbre complet de résolution de but : manger(X,Y).
3.

Solution

1.

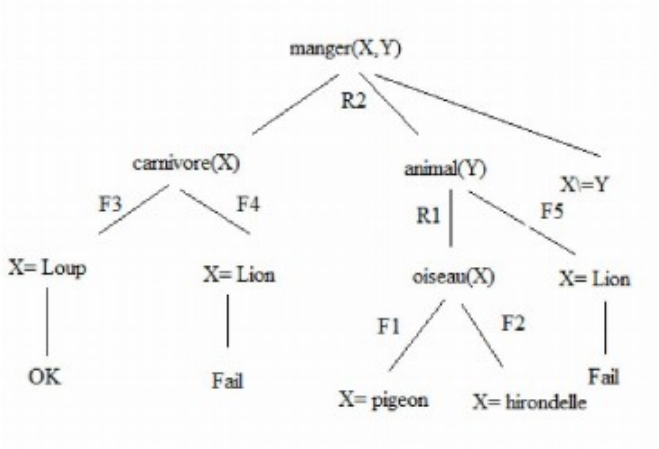
F1 oiseau(pigeon).
F2 oiseau(hirondelle).
F3 carnivore(loup).
F4 carnivore(lion).
F5 animal(lion).
R1 animal(X) :-oiseau(X).
R2 manger(X,Y) :-carnivore(X),animal(Y), X\=Y.

1. Par unification : manger(lion, Y) = manger(X,Y) / X= lion, Y = ?
2. On applique R2 : on doit vérifier carnivore(lion) et animal(Y), Y \= lion.
3. A partir F4 : carnivore(lion) est vrai il reste à vérifier animal(Y) tel que Y \= lion.
4. A partir R1 : on doit vérifier oiseau(X). A partir F1 et F2 X= pigeon et X= hirondelle.

Donc :

Y = pigeon ;
Y = hirondelle.

2. Arbre de résolution



Exercice 13

homme(Ahmed). homme(Ali). homme(Mohammed). homme(amine). homme(ibrahim). homme(ishak). femme(imen). femme(hesna).	femme(amina). femme(maria). femme(fatima). pere(Ahmed,Mohammed). pere(Mohammed,Ali). pere(Ali,amine). pere(Ali,fatima).	pere(Mohammed,amina). pere(ibrahim,ishak). mere(imen,Mohammed). mere(hesna,amina). mere(hesna,Ali). mere(amina,ishak). mere(maria,amine). mere(maria,fatima).

1.Traduire les questions suivantes en Prolog et vérifier les réponses :
Est-ce que Ali est un homme ?
Est-ce que ishak est une femme ?
Qui est une femme ?
Qui est un homme ?
Est-ce que maria est la mère de fatima ? De Ishak
Qui est la mère de Mohammed ?
Quels sont les enfants de Ali ?
Quels sont les hommes qui sont pères ?

2. Définir les prédicats suivants :
parent(X,Y) : X est un parent de Y, pere ou mere
fils(X,Y) : X est le fils de Y ;
fille(X,Y) : X est la fille de Y ;
grand_pere(X,Y) : X est le grand-père de Y ;
grand_mere(X,Y) : X est la grand-mère de Y ;
frere(X,Y) : X est le frère de Y ;
soeur(X,Y) : X est la soeur de Y.

4. Traduction d'énoncés
Traduire en Prolog l'énoncé suivant :
maria aime le coca
mourad est un voleur.
mourad aime tous ceux qui aiment le coca.
Si quelqu'un est un voleur et aime quelque chose alors il le vole.
Qui vole quoi?

Exercice 14 (Prolog et la récursivité)
1. Dire si un nombre est pair.

```
Pair(0).
pair(X) :- X>0, X2 is X-2, pair(X2).
```

2. Trouver la somme des **N** premiers entiers. (Ou : **som(N,X)** est vrai si **X** est la somme des entiers de **1** à **N**.)

```
som(0,0).
som(N,X) :- N>0, N1 is N-1, som(N1,X1), X is N+X1.
```

3. Trouver la factorielle d'un nombre. (Ou : **fact(N,X)** est vrai si **X** vaut **N!**.)

```
fact(0,1).

fact(N,X) :- N>0, N1 is N-1,

fact(N1,X1), X is N*X1.
```

4. **fibo(N,X)** est vrai si **X** est la valeur de la suite de *Fibonacci* au rang **N**.

```
fibo(1,1).
 Fibo(2,1).
 fibo(N,X) :- N>2, U is N-1, V is N-2, fibo(U,U1), fibo(V,V1),
                  X is U1+V1.
```

Exercice 14

- Adam aime les pommes.
- Amina aime les carottes.
- Omar aime les oranges.
- Les pommes sont des fruits.
- Les oranges sont des fruits.
- Les carottes sont des légumes.

Ceux qui aiment les fruits sont en bonne santé.

1. Formalisez ces faits et règles en PROLOG.
2. Quelle est la requête pour savoir qui est en bonne santé ?
3. Quelle est la requête pour “Qui aime les pommes?”
4. Comment savoir les fruits que connaît le programme ?