Introduction à l'intelligence artificielle

Introduction à l'intelligence artificielle:

- A. Histoire de l'IA
- B. Nature de l'IA

Histoire de l'intelligence artificielle

L'histoire de l'intelligence artificielle a passé par plusieurs étapes nous citons ici les étapes les plus marquantes:

Gestation de l'IA (1943-1955) durant cette étape les premiers essais ont été faits. Ces essais peuvent être considérés comme le début de l'IA. Les travaux les marquants dans cette étape sont :

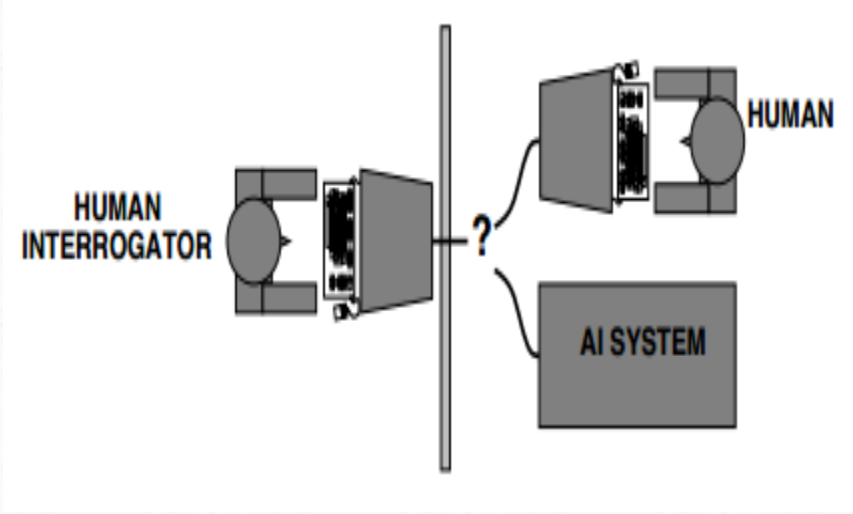
- L'introduction d'un modèle de neurones artificiels par McCulloch et Pitts en 1943.
- Hebb proposa une règle pour modifier des connexions entre neurones
- Minsky et Edmonds ont construit le premier réseau de neurones

Histoire de l'intelligence artificielle

- En 1953 Turing a publié son fameux test de Turing qui consiste à:
- un jeu avec 3 joueurs A, B et C
  C doit deviner qui entre A et B est un homme et qui est une femme
  C pose des questions à A et B
  le joueur A tente d'induire C en erreur
  le joueur B joue le jeu

Histoire de l'intelligence artificielle

Que se passe-t-il quand on remplace A par une machine ?
C'échouera-t-il aussi souvent ?



Histoire de l'intelligence artificielle

Naissance d'IA (1956)

Pendant cette année un petit groupe d'informaticiens passionnés par l'étude de l'intelligence ont été réunis dans une conférence. Durant cette conférence ils ont posé les fondements de l'intelligence artificielle.

Espoirs grandissants (1952-1969)

Cette période a été très active où un nombre très important de programmes ont été créés pour résoudre divers problèmes. Les événements les plus marquants sont:

- Les programmes Logic Theorist (Newell et Simon) et Geometry Theorem Prover (Gelernter) visant de prouver certains théorèmes mathématiques (1956).

Histoire de l'intelligence artificielle

- Le General Problem Solver de Newell et Simon réussissait à résoudre des problèmes basiques avec un raisonnement semblable à celui de l'être humain (1957).
- Samuel a créé un programme qui joue aux dames (1957).
- Des étudiants de Minsky qui travaillent sur les petits problèmes «microworlds», ce qui a donné naissance au programme ANALOGY.
- McCarthy a publié un article où il traite des programmes qui ont du sens commun.

Histoire de l'intelligence artificielle

- La recherche sur les réseaux de neurones a poursuivi.
- La création de Shakey, le premier robot qui a été capable de raisonner sur ses propres actions.

Histoire de l'intelligence artificielle

Premières Deceptions (1966-1973)

Pendant ces années les prédictions des chercheurs de l'IA ont été très optimistes. Par exemple, les chercheurs n'ont compté que 5 ans pour créer un traducteur automatique. Après, ils sont rendus compte que leur approche basée que sur le côté syntaxique n'est pas suffisante. Suite à cet échec, le gouvernement américain a annulé en 1966 tout financement pour les projets de traduction automatique.

Histoire de l'intelligence artificielle

Une deuxième grande déception est apparue lorsque les chercheurs de l'IA ont testé leurs algorithmes sur des problèmes de grande taille. Ils ont découvert que leurs algorithmes ne fonctionnent pas à cause la faible puissance de calcule et le manque de mémoire. Après cet échec et suite aux critiques adressés à l'IA dans le rapport de Lighthill en 1973 la Grande-Bretagne a arrêté le financement de la quasi-totalité des projets en IA

Histoire de l'intelligence artificielle

En 1969 la branche de l'apprentissage automatique de l'IA a connu une grande crise suite à la publication du livre "Perceptrons" de Minsky et Papert, qui ont prouvé que les réseaux de neurones de leur époque ne pouvaient pas calculer certaines fonctions très simples.

Histoire de l'intelligence artificielle

Systèmes Experts (1969-1979)

En 1969 le premier système expert, appelé DENDRAL a été créé afin de déterminer la structure moléculaire d'une molécule. DENDRAL est basé sur un grand nombre de règles heuristiques élaborées par des experts humains.

Suite au succès du DENDRAL, d'autres systèmes d'experts ont été créés, comme le fameux MYCIN, qui réalisait un diagnostic des infections sanguines.

Histoire de l'intelligence artificielle

L'IA dans l'Industrie (1980-présent)

L'entreprise DEC au début des années 80 a utilisé un système expert qui l'aide à configuré des systèmes informatiques, ce qui lui a permet de faire beaucoup d'économie (quelques millions de dollars).

Suite à ce succès, plusieurs grandes entreprises ont commencè à s'intéresser à l'IA et a créé leur propre labo de recherche.

Les États-Unis et la Grande-Bretagne ainsi que le Japon financèrent de gros projets en IA.

Histoire de l'intelligence artificielle

Le retour des réseaux de neurones (1986-présent)

Au milieu des années 80, des chercheurs ont découvert la règle d'apprentissage "back-propagation" qui a permis de créer des réseaux de neurones capables d'apprendre des fonctions très complexes.

Après cette découverte l'apprentissage automatique est devenu l'un des domaines les plus actifs de l'IA, et il a été appliqué à de nombreux problèmes comme la fouille de données (Data Mining)

Histoire de l'intelligence artificielle

L'IA adopte les méthodes scientifiques (1987-présent)

L'intelligence artificielle est devenue de plus en plus rigoureuse et formelle. Actuellement la plupart des approches étudiées sont basées sur des fondements mathématiques ou des études expérimentales au lieu de l'intuition.

L'IA est appliquée souvent à des problèmes issus du monde réel.

Histoire de l'intelligence artificielle

Émergence des agents intelligents (de 1995- à nos jours):

Encouragés par les progrès dans la résolution de sous-problèmes de l'IA, les chercheurs ont également reconsidéré le problème de l'«agent total ».

Le travail d'Allen Newell, John Laird, et Paul Rosenbloom sur SOAR est l'exemple le plus connu d'une architecture d'agent complet. L'un des environnements les plus importants pour les agents intelligents est l'Internet.

Tellement les systèmes d'IA sont devenus si courants dans le Web, le suffixe « -bot » est devenu très utilisé dans le langage courant.

En outre, de nombreux outils Internet, tels que les moteurs de recherche, les systèmes de recommandation, et les agrégateurs des sites Web reposent sur des techniques de l'IA.

Histoire de l’intelligence artificielle

La disponibilité de vastes ensembles de données (de 2001 à nos jours): Durant 60 ans de l'histoire de l'informatique, les travaux ont focalisé sur les algorithmes comme sujet principal de l'étude. Mais certains travaux récents en IA suggère que pour de nombreux problèmes, il est plus logique de se focaliser sur les données et d'être moins pointilleux sur ce que l'algorithme à appliquer.

Cela est devenu une réalité en raison de la disponibilité croissante de très grandes sources de données: par exemple, des milliards de mots d'anglais et des milliards d'images à partir du Web.

Nature de l'intelligence artificielle

Qu'est-ce que l'Intelligence ?

Il existe plusieurs définitions de l'intelligence:

Wikipédia: Le terme est dérivé du latin intellegentia, « faculté de comprendre », dont le préfixe inter- (« entre »), et le radical legere (« choisir, cueillir ») ou ligare (« lier ») suggèrent essentiellement l'aptitude à lier des éléments entre eux.

Larousse: l'intelligence est la faculté de comprendre, de saisir par la pensée.
l'intelligence est l'aptitude à s'adapter à une situation, à choisir en fonction des circonstances.

Nature de l'intelligence artificielle

Autres définitions de l'intelligence:

- Ensemble des fonctions mentales ayant pour objet la connaissance conceptuelle et rationnelle (par opposition à la sensation et à l'intuition). Aptitude à comprendre et à s'adapter facilement à des situations nouvelles.) D'après Louise Bérubé (neurologue).
- L'intelligence est la capacité de comprendre un contexte nouveau, de le comprendre et de réagir à cette nouvelle situation de façon adaptée. D'après Richard Atkinson (Président Université de Californie)
- L'intelligence, c'est ce qui permet d'entendre une musique là où d'autres n'entendent qu'un bruit. D'après Jean-Charles Terrassier (Psychologue-Pédiatre, spécialiste des surdoués)

Nature de l'intelligence artificielle

- L'intelligence, ça n'est pas ce que l'on sait, mais ce que l'on fait quand on ne sait pas. D'après Jean Piaget (Psychologue, Biologiste, Logicien).
- Quel truc magique nous rend intelligents ? Le truc, c'est qu'il n'y a pas de truc. La puissance de l'intelligence provient de notre vaste diversité, et non d'un seul principe parfait. D'après Marvin Minsky.

Nature de l'intelligence artificielle

Quelques exemples de tâches"intelligentes :

- le raisonnement dans bon sens.
- l'étude des sciences comme la physique, mathématiques . . . etc.
- Le Cultivassions de la terre.
- La domestication des animaux.
- La compréhension des langues.
- Le développement des logiciels.
- La conduite d'un véhicule.
- la traduction.

Nature de l'intelligence artificielle

- **Les différentes formes de l'intelligence**
- **L'intelligence logico-mathématique**, liée à la vitesse à laquelle on peut résoudre un problème numérique ou logique.
- **L'intelligence linguistique**, fréquente chez les politiciens, écrivains, poètes...
- **L'intelligence intrapersonnelle**, désigne la capacité qu'on a pour avoir une vision critique sur soi-même, ses limites, ses réactions...
- **L'intelligence interpersonnelle**, grâce à laquelle on peut deviner les réactions de son entourage.
- **L'intelligence visuospatiale** permet de manipuler des objets tridimensionnels dans sa tête. Architectes, géographes, artistes...

Nature de l'intelligence artificielle

- **L'intelligence naturaliste**, permets de classer les objets, et de les différencier en catégories. Zoologistes, botanistes, archéologues...
- **L'intelligence musicale** juge la hauteur, la tonalité des sons, le rythme et la mélodie d'une musique. Musiciens, compositeurs...
- **Éventuellement l'intelligence existentialiste ou spirituelle**, aptitude à se questionner sur le sens et l'origine des choses.

Nature de l'intelligence artificielle

Qu'est-ce que l'Intelligence artificielle ?

il n' y a pas de consensus sur la définition du terme "intelligence artificielle ", voici quelques définitions qu'on peut trouver dans la littérature:

"l'étude des facultés mentales à l'aide des modèles de type calculatoires " (Charniak et McDermott)

"conception d'agents intelligents" (Poole)

"discipline étudiant la possibilité de faire exécuter par l'ordinateur des taches pour lesquelles l'homme est aujourd'hui meilleur que la machine" (Rich et Knight)

"l'automatisation des activités associées au raisonnement humain, telles que la décision, la résolution de problèmes , l'apprentissage ..." (Bellman)

"l'étude des mécanismes permettant à un agent de percevoir, raisonner, et agir" (Winston)

"l'étude des entités ayant un comportement intelligent" (Nilsson)

Nature de l'intelligence artificielle

ces différentes définitions s'accordent sur le fait que l'objectif de l'IA est de créer des systèmes intelligents, par contre on distingue deux types de discords.

1- la focalisation sur:

- le comportement du système.
- Le fonctionnement interne (le raisonnement) du système.

2- font la liaison avec:

- l'intelligence de l'être humain.
- Un standard de rationalité plus général (ne font pas référence aux humains).

Nature de l'intelligence artificielle

On peut distinguer quatre façons de voir l'intelligence artificielle:

- *créer des systèmes qui se comportent comme les êtres humains* - cette définition est soutenue par Alan Turing dans son "test de Turing" , donc 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.
- *Créer des systèmes qui pensent comme des êtres humains* - cela implique que l'IA est une science expérimentale, donc il faut comprendre la façon dont pensent les êtres humains, ensuite il faut évaluer les systèmes par rapport au raisonnement de l'être humain.

Nature de l'intelligence artificielle

- *créer des systèmes qui pensent rationnellement* - donc le système doit suivre les lois de la logique. Cette approche peut être critiquée, car il semble que certaines capacités ne sont pas facilement exprimables en logique.

- *Créer des systèmes qui possèdent des comportements rationnels* - cette définition de l'IA concerne le développement des agents qui agissent pour mieux satisfaire leurs objectifs.

Nature de l'intelligence artificielle

Donc l'IA peut être résumée par la création:

Des systèmes qui pensent comme des humains	Des systèmes qui pensent de manière rationnelle
Des systèmes qui agissent comme des humains	Des systèmes qui agissent de manière rationnelle

Que peut-on faire avec l'IA ?

Jouer correctement au tennis de table ?
Oui , le robot d'Anderssone (1988)

Conduire en toute sécurité à Alger dans une artère bondée ?
Non pas encore, mais des gens y travaille cf. Darpa Project

Commander sur le web une semaine de nourriture ?
Oui, sans aucun problème et au meilleur prix

Faire les courses de la semaine dans un centre commercial ?
Non, pas encore malheureusement, sauf pour des produits limites

Nature de l'intelligence artificielle

Donner un conseil juridique avise
Oui, dans des domaines spécialistes du droit

Traduire de l'anglais au français en temps réel ?
Oui, pour des textes limites

Discuter pendant une heure avec un être humain ?
Non, mais je peux faire illusion quelques instants.

Réaliser une opération chirurgicale complexe ?
Oui, mais sous surveillance d'un chirurgien

Vider un lave-vaisselle et tout ranger ?
Non, pas encore malheureusement.

Nature de l'intelligence artificielle

Découvrir et prouver un nouveau théorème mathématique ?
Oui, pour certains, mais pas pour tous

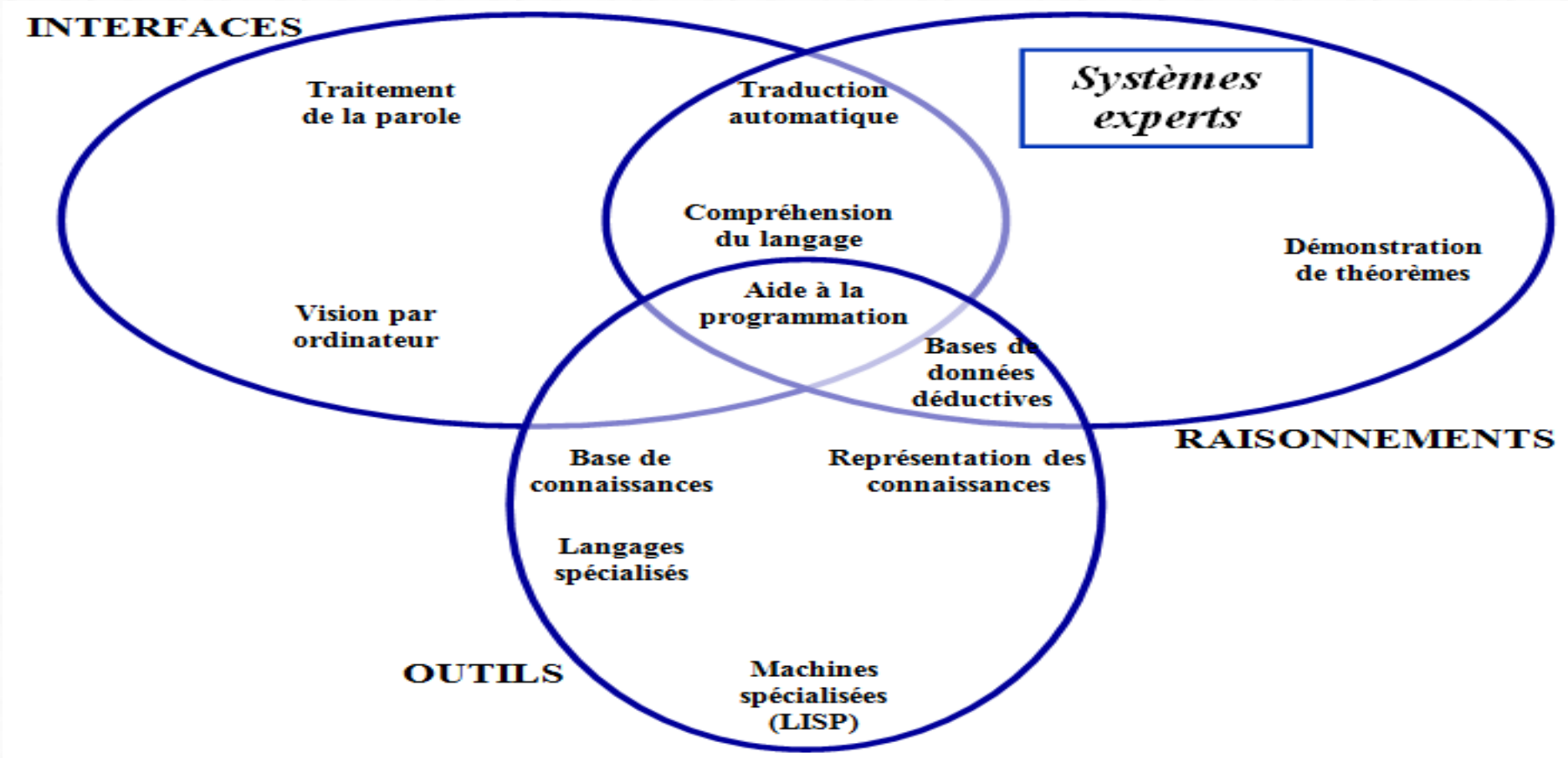
Concevoir et réaliser un programme de recherche en biologie moléculaire ?
Non, pas encore

Écrire une histoire drôle
Non, pas intentionnellement

Tondre la pelouse ?
Oui, sans problème

Nature de l'intelligence artificielle

Les domaines de l'intelligence artificielle



Nature de l'intelligence artificielle

Exemple de quelques domaines de l'IA

- **Représentation des connaissances et Raisonnement automatique**

Il vise le traitement des problèmes de la représentation des connaissances (qui peuvent être incomplètes, incertaines, ou incohérentes) et de la mise en œuvre du raisonnement.

- **Résolution de problèmes généraux:** son objectif est de créer des algorithmes généraux pour résoudre des problèmes concrets.
- **Traitement du langage naturel:** son but est la compréhension, la traduction, ou la production du langage (écris ou parle).
- **Vision artificielle:** son objectif est de permettre aux ordinateurs de comprendre les images et la vidéo (par exemple, la reconnaissance des formes).

Nature de l'intelligence artificielle

- **Robotique:** son objectif est de réaliser des agents physiques qui peuvent agir dans le monde.
- **Apprentissage automatique:** vise de concevoir des programmes qui peuvent s'auto-modifier en fonction de leur expérience.

Nature de l'intelligence artificielle

Quelques succès de l'IA

- Deep blue [Jeu d'échecs]
- Mogo [Jeu de Go]
- ViaVoice [Reconnaissance vocale]
- Codage postal [Reconnaissance d'écriture]
- Watson [Système expert]
- Aibo et Asimo [Robots de compagnie]
- Robots assistants [Robots médicaux]
- Bigdog [Robots militaires]
- Google Driverless, Car [Véhicule autonome]

Représentation des connaissances

Représentations logiques

Introduction

Qu'est-ce qu'une connaissance ?

Une connaissance c'est le *fait* ou la *condition* de connaître quelque chose avec l'expérience.

Qu'est-ce qu'une représentation ?

La représentation est l'action de rendre sensible un concept au moyen d'une figure, d'une écriture, d'un langage ou d'un formalisme.

Introduction

Les connaissances doivent être représentées:

- D'une manière efficace (malgré qu'il est impossible de représenter explicitement chaque élément).
- Avec sens (comment relier les connaissances avec le monde réel).

Introduction

Qu'est-ce que la représentation des connaissances

- La représentation des connaissances, comme son nom l'indique, a pour objectif l'étude des formalismes qui permettent la représentation de toutes formes de connaissances.

- La correspondance entre les caractéristiques du monde et le langage formel (sémantiques des langages de représentation).

Introduction

Problématique:

- Comment rendre l'ordinateur « au courant » de quelque chose ?
- C'est le problème de la *représentation des connaissances* afin de pouvoir la stocker d'une manière intelligible dans la mémoire de l'ordinateur.

Introduction

Parmi les techniques de représentation des connaissances, on peut citer:

- **La représentation relationnelle**
- **La représentation hiérarchique**
- **La représentation mathématique :** utilisation la **logique mathématique**: proche du langage naturel**.**
- **La logique des propositions :** un énoncé ou un fait est une variable à deux valeurs: vrai ou faux. Les connecteurs logiques sont utilisés.

Introduction

- **logique des prédicats du premier ordre:** fonction ayant un ou plusieurs paramètres, prenant la valeur vraie ou fausse selon le(s) paramètre(s).
- **Les règles de production :** quantité de connaissance sous la forme de:
  SI condition ALORS action.
- **Les réseaux sémantiques :** ensemble de nœuds (concepts) reliés par des arcs (relations entre les concepts).
- **Les ontologies**

…….

Représentation logique

Originaires des développements théoriques dans le domaine de la logique formelle, ces représentations remontent aux premiers jours de l'IA (Logique Theorist de Newell, Shaw et Simon en 1956).

Comme modèle de la représentation déclarative, elles concernent surtout la logique mathématique (logique des propositions et logique des prédicats du premier ordre).

Mais d'autres logiques non standard sont également utilisées.

La logique propositionnelle

- La logique propositionnelle est une logique très simple sur laquelle se basent presque toutes les logiques étudiées aujourd'hui.
- Les éléments de base sont des *propositions* (ou variables propositionnelles) qui représentent des énoncés qui peuvent être soit vrais soit faux dans une situation donnée.

La logique propositionnelle

- Dans la logique propositionnelle, les éléments de base sont des *propositions* ou variables propositionnelles qui représentent des énoncés qui peuvent être soit vrais soit faux dans une situation donnée.

Par exemple:

p est une proposition qui représente “Ali est un étudiant”.

q est une proposition qui représente l’énoncé “Deux est un nombre pair”.

La logique propositionnelle

Des formules logiques composées (ou formule bien formée) peuvent être construites à partir des propositions en utilisant les connectives logiques : $\wedge$(“et”), $\vee$ (“ou”), e t $\neg$ (négation), $\Rightarrow$ (l’implication logique), $\Leftrightarrow$(l’équivalence).

$p \Rightarrow q$ est une abréviation pour $\neg p \vee q$

$p \Leftrightarrow q$ est une façon concise d’écrire $(p \wedge q) \vee (\neg p \wedge \neg q)$.

La logique propositionnelle

Exemple:
p1 = “il pleut”,
p2 =“il fait beau”,
p3 = “l’herbe est mouillée”,
p4 = “après le repas, je tonds la pelouse”,
p5 = “Il y a un bon film à la télévision ce soir”

P6= “il pleut cet après-midi”

P7= “ l’herbe est mouillée”

¬ (non : 1) ¬p4 = “Je **ne** tonds **pas** la pelouse après le repas”
∧ (et : 2) p1 ∧ p5 = “Il pleut **et** il y a un bon film à la télévision ce soir”
∨(ou : 2) p1 ∨ p2 = “Il pleut **ou** il y a un bon film à la télévision ce soir”
⇒ (implique) p6 ⇒ p7 = “s’il pleut cet après-midi, l’herbe sera mouillée”
⇔ (équivaut) p1 ⇔ ¬p2 = “il pleut si et seulement si il ne fait pas beau”

La logique propositionnelle

Il est parfois nécessaire d'insérer des parenthèses pour définir des priorités entre ces différents connecteurs. L'ordre de priorité de ces connecteurs est donné par l'ordre décroissant suivant: $\Leftrightarrow$, $\Rightarrow$, $\wedge$, $\vee$, $\neg$.

Lorsque le même connecteur se répète, la priorité est de gauche à droite.

Exemple :

$P \Leftrightarrow Q \wedge R$ *est équivalente à* $(P \Leftrightarrow (Q \wedge R))$

$P \Rightarrow Q \wedge \neg R \vee S$ *est équivalente à* $(P \Rightarrow (Q \wedge ((\neg R) \vee S)))$

La logique propositionnelle

L'ensemble des propositions bien formées (pbf) est le plus petit ensemble tel que :

– un atome(proposition élémentaire) est une pbf
– si P, P1 et P2 sont des pbf, alors
– ¬P est une pbf
– (P1 ∧ P2) est une pbf
– (P1 ∨ P2) est une pbf
– (P1 ⇒ P2) est une pbf
– (P1 ⇔ P2) est une pbf

La logique propositionnelle

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
faux	*faux*	*vrai*	*faux*	*faux*	*vrai*	*vrai*
faux	*vrai*	*vrai*	*faux*	*vrai*	*vrai*	*faux*
vrai	*faux*	*faux*	*faux*	*vrai*	*faux*	*faux*
vrai	*vrai*	*faux*	*vrai*	*vrai*	*vrai*	*vrai*

La logique propositionnelle

Une proposition élémentaire s'appelle un **atome**.

Deux propositions atomiques sont soit identiques (deux noms pour la même proposition), soit sans rapport.

Une proposition qui est soit un atome, soit la négation d'un atome s'appelle un **littéral**.

Une *clause* est une disjonction de littéraux

Forme normale conjonctive (fnc) ou forme clausale : conjonction de clauses.

Forme normale disjonctive (fnd) : disjonction de conjonction d'atome. C'est la duale de la fnc.

La logique propositionnelle

- $(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$ implication élimination
- $(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ élimination bi-conditionnelle
- $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ De Morgan
- $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ De Morgan
- $(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributivité de $\wedge$ sur $\vee$
- $(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributivité de $\vee$ sur $\wedge$

La logique propositionnelle

- $(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativité de $\wedge$
- $(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ commutativité de $\vee$
- $((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativité de $\wedge$
- $((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ associativité de $\vee$
- $\neg(\neg\alpha) \equiv \alpha$ double-négation élimination
- $(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ contraposition

La logique propositionnelle

Mise en forme Clausale :

- Etape1 : $\phi \Leftrightarrow \psi \rightarrow (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$

  $\phi \Rightarrow \psi \rightarrow \neg\phi \vee \psi$

- Etape2 $\neg\neg\phi \rightarrow \phi$

  $\neg(\phi \wedge \psi) \rightarrow \neg\phi \vee \neg\psi$

  $\neg(\phi \vee \psi) \rightarrow \neg\phi \wedge \neg\psi$

- Etape3

  $\phi \vee (\psi 1 \wedge \psi 2) \rightarrow (\phi \vee \psi 1) \wedge (\phi \vee \psi 2)$

  $(\psi 1 \vee \psi 2) \wedge \phi \rightarrow (\psi 1 \vee \phi) \wedge (\psi 2 \vee \phi)$

La logique propositionnelle

Un *modèle M* est une fonction qui donne une valeur de vérité (soit *vrai*, soit *faux*) à chaque proposition du langage.

Une proposition *p* est satisfaite dans un modèle *M* si elle reçoit la valeur *vrai donc on écrit* $M \models P$

Une proposition *p* n'est pas satisfaite dans un modèle si elle reçoit la valeur *faux donc on écrit* $M \not\models P$

La logique propositionnelle

La *satisfaction* des formules dans des modèles est entièrement déterminée par les valeurs de vérité des propositions de la formule :

$M \models \phi \wedge \psi$ si et seulement si $M \models \phi$ et $M \models \psi$

$M \models \phi \vee \psi$ si et seulement si $M \models \phi$ ou $M \models \psi$ (ou les deux)

$M \models \neg\phi$ si et seulement si $M \not\models \phi$

La logique propositionnelle

- Une formule est *valide* si elle est satisfaite par tout modèle
- Une formule est *satisfiable* si elle est satisfaite dans au moins un modèle.
- Une formule ϕ est une *conséquence logique* de ψ (ce que l'on note $\phi \models \psi$) si chaque modèle qui satisfait ϕ satisfait aussi ψ , ou autrement dit, s'il n'y a pas de modèle de $\phi \wedge \neg \psi$.
- $\alpha \equiv \beta$ si et seulement si $\alpha \models \beta$ and $\beta \models \alpha$

La logique propositionnelle

- **Insuffisance de la logique des propositions:** La logique des propositions se révèle insuffisante lorsque l'on veut déduire des propriétés valables pour des ensembles d'éléments du monde réel.

- La logique des prédicats du premier ordre fournit les moyens de préciser la portée d'assertions plus générales et donc d'exprimer ce type de connaissance.

Représentation des connaissances

Représentations logiques

La logique du premier ordre

La logique du premier ordre est une logique très expressive et bien étudiée.

Les formules en logique du premier ordre sont formée des éléments suivants :

- – les symboles pour constants, e.g. Marie, gareMarseille
- – les variables, e.g. x, y
- – les symboles pour fonctions, e.g. méreDe
- – les symboles pour prédicats, e.g. humain, dans

La logique du premier ordre

- les connectives logiques : $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$.
- le quantificateur universel $\forall$
- le quantificateur existentiel $\exists$
- le symbole d'´egalite $=$

La logique du premier ordre

L'arité détermine combien d'arguments peut prendre chaque symbole pour fonction et chaque symbole pour prédicat

e.g. *mèreDe* a un arité de 1.

Les termes sont des symboles pour constantes, variables, ou bien des fonctions appliquées à d'autres termes,

e.g. Marie, mèreDe(x), mèreDe(mèreDe(Marie))

La logique du premier ordre

Les atomes sont de deux types :

- $P(terme_1,..., terme_n)$ où P est un symbole pour prédicat avec arité n et les $terme_i$ sont des termes

e.g. humain(Marie), dans(Marie, gareMarseille)

- $terme_1 = terme_2$ où $terme_1$ et $terme_2$ sont des termes.

e.g. Marie = mèreDe(x)

La logique du premier ordre

Les formules sont:

- des atomes, e.g. humain(Marie),
- des combinaisons booléennes de formules, e.g. dans(Marie, gareMarseille) ∧ humain(Marie) ,
- Les formules préfixées par ∃ ou ∀ (où x est un variable), e.g. ∃ x humain(x), ∀ x humain(x).

La logique du premier ordre

Une Formule Atomique

Une formule atomique est représentée par p(t_1, ..., t_n) où p est un symbole de prédicat et les ti des termes. Un prédicat est utilisé en général pour représenter une relation dans un certain contexte.

Exemple1 *:* La phrase *Ali est allé à l'université* peut être représentée par :

est_*allé*_*à*(*Ali, université*)

Un littéral: Un littéral est défini comme une formule atomique ou sa négation

La logique du premier ordre

Formule Bien Formée (fbf):

- Toute formule atomique est une fbf,
- Toute conjonction de fbf est une fbf,
- Toute disjonction de fbf est une fbf,
- La négation d'une fbf est une fbf,
- Si p et q sont deux fbfs alors p ⇒ q est une fbf,
- Si p et q sont deux fbfs alors p ⇔ q est une fbf.

La logique du premier ordre

- **Propriétés des fbf**
- Soient X1 et X2 2 fbfs quelconques, on peut établir les relations suivantes :
- $\neg(\neg X1) \equiv X1$
- $X1 \Rightarrow X2 \equiv \neg X1 \vee X2$
- **Loi de De Morgan**
- $\neg(X1 \wedge X2) \equiv \neg X1 \vee \neg X2$
- $\neg(X1 \vee X2) \equiv \neg X1 \wedge \neg X2$
- **Lois distributives**
- $X1 \wedge (X2 \vee X3) \equiv (X1 \wedge X2) \vee (X1 \wedge X3)$
- $X1 \vee (X2 \wedge X3) \equiv (X1 \vee X2) \wedge (X1 \vee X3)$

La logique du premier ordre

- **Lois commutatives**
- $X1 \wedge X2 \equiv X2 \wedge X1$
- $X1 \vee X2 \equiv X2 \vee X1$
- $(X1 \wedge X2) \wedge X3 \equiv X1 \wedge (X2 \wedge X3)$
- $(X1 \vee X2) \vee X3 \equiv X1 \vee (X2 \vee X3)$
- **Loi de Contraposition**
- $X1 \Rightarrow X2 \equiv \neg X2 \Rightarrow \neg X1$

La logique du premier ordre

- **Autres Équivalences:**
- $\neg [(\exists x)P(x)] \equiv (\forall x) \neg P(x)$
- $\neg [(\forall x) P(x)] \equiv (\exists x) \neg P(x)$
- $(\forall x)[P(x) \wedge Q(x)] \equiv (\forall x)P(x) \wedge (\forall y) Q(y)$
- $(\exists x)[P(x) \vee Q(x)] \equiv (\exists x)P(x) \vee (\exists y)Q(y)$
- $(\forall x)P(x) \equiv (\forall y)P(y)$
- $(\exists x)P(x) \equiv (\exists y)P(y)$

La logique du premier ordre

- Si une fbf a la valeur de vérité Vraie (respectivement Fausse) pour toutes les interprétations possibles, alors elle est dite Valide (respectivement inconsistante).
- Les fbf (sans variables) valides sont appelées tautologies.
- Une fbf X est une conséquence logique d'un ensemble S de fbf si toute interprétation satisfaisant S satisfait X.

La logique du premier ordre

- Un modèle en logique du premier ordre est compose d'un ensemble non vide U (appelé l'univers) et d'une fonction d'interprétation I.

La fonction I associe :

- à chaque symbole pour constante C un élément de l'univers $I(C) \in U$,
- à chaque symbole pour fonction F (où l'arité de F est n) une fonction n-aire de U à U,
- à chaque symbole pour prédicat P (où l'arité de P est n) un prédicat n-aire sur U.

La logique du premier ordre

- Les valuations sont des fonctions qui traitent les variables (qui ne sont pas fixés par des modèles), où ils associent à chaque variable un membre de l'univers U.

- Dans la définition de la satisfaction, on a besoin de la notation v(x/d), qui donne la valuation qui est la même que v sauf pour la variable x, à laquelle on associe l'élément de l'univers d.

La logique du premier ordre

La satisfaction d'une formule à par rapport un modèle M =< U, I > et une valuation v comme ceci :

- $\models_{M,v} t1 = t2$ ssi $t1$ et $t2$ réfère au même élément de U
- $\models_{M,v} P(t1,...,tn)$ ssi le tuple d'éléments associes à $(t1,...,tn)$ est dans $I(P)$
- $\models_{M,v} \phi \wedge \psi$ ssi $\models_{M,v} \phi$ et $\models_{M,v} \psi$

La logique du premier ordre

- $\models_{M,v} \phi \vee \psi$ ssi $\models_{M,v} \phi$ ou $\models_{M,v} \psi$
- $\models_{M,v} \neg \phi$ ssi $\not\models_{M,v} \phi$
- $\models_{M,v} \forall x.\ \phi$ ssi pour tout $d \in U$ nous avons $\models_{M,v(x/d)} \phi$
- $\models_{M,v} \exists x.\ \phi$ ssi il existe $d \in U$ tel que $\models_{M,v(x/d)} \phi$

La logique du premier ordre

- Une formule est valide si elle est satisfaite par tout mode le et toute valuation.

- La logique du premier ordre est beaucoup plus puissante que la logique propositionnelle, et cette expressivité accrue se paie par une difficulté du calcul.

La logique du premier ordre

Règles d'inférence : Dans ce langage des prédicats, il existe des règles d'inférence qui peuvent être appliquées à certaines fbf pour produire de nouvelle fbf.

Exemples de règle d'inférence:

- Le Modus Ponens: A partir de W1 et W1 ⇒ W2 on infère W2
- La Spécialisation universelle: A partir de (∀ x)W(x) on infère W(A) où A est un symbole de constante.
- Combinaison des 2 règles précédentes: A partir de (∀ x)[W1(x) ⇒ W2(x)] et W1(A) on infère W2(A)

Les fbf inférées sont appelées des théorèmes. La séquence d'application des règles d'inférence utilisées dans la dérivation constitue la démonstration de ce théorème.

La logique du premier ordre

- **Système de Règles d'inférence sain :** Un système de règles d'inférence est dit sain si tout théorème dérivable de tout ensemble de fbf est conséquence logique de cet ensemble de fbf. On peut démontrer que le Modus Ponens est sain.
- **Système de Règles d'inférence complet :** Un système de règles est complet si toutes les fbf qui découlent logiquement de tout ensemble de fbf sont aussi des théorèmes dérivables de cet ensemble

La logique du premier ordre

- **Unification :** Dans la règle d'inférence combinée précédente il a été nécessaire de trouver la substitution « x prend la valeur A » pour rendre W1(x) et W1(A) identique. Ce processus est appelé *Unification*. Il existe des algorithmes pour effectuer cette unification.

La logique du premier ordre

Définition de la substitution: Une substitution σ est une application

σ: terme ----> terme

qui est l'identité sauf en un certain nombre de point de variable. Elle est notée par l'ensemble {<Xi/ti>}. Xi est une variable, ti est un terme. (on la note des fois par <Xi,ti>

- L'instanciation t' d'un terme t est définie comme étant l'application d'une substitution σ à t. On la note par t' = σ(t).

La logique du premier ordre

- **Exemple :** Soit la fbf P(x, f(y), B) où x, y sont des variables, B est une constante et f une fonction. On peut obtenir les instances suivantes:
  - P(z, f(w), B) par la substitution {x/z, y/w} (Renommage)
  - P(x, f(A), B) par la substitution {y/A}
  - P(C, f(A), B) par la substitution {x/C, y/A} (appelée Instance Close - pas de variable)

La logique du premier ordre

- **Composition de substitution :** La composition de substitution est définie comme suit : $\sigma1.\sigma2(p) = \sigma2(\sigma1(p))$

- **Définition** termes t1 et t2 sont unifiables s'il existe une substitution σ telle que $\sigma(t1)=\sigma(t2)$.

La logique du premier ordre

Algorithme UNIFICATEUR(S)

1. k=1; $\sigma_1 = \varepsilon$
2. **Si** σ_k est unificateur pour S,
   Alors retourner σ_k comme UPG de S
   Sinon calculer D_k l'ensemble de désaccord de $S\sigma_k$
3. **Si** il existe une paire (v, t) telle que v est une **variable** dans D_k qui n'apparaît pas dans t et $\{v = t\}$ est un unificateur pour D_k,
   alors $\sigma_{k+1} = \sigma_k\{v = t\}$, k=k+1;
   retourner à 2.
   Sinon exit; S n'est pas unifiable.

La logique du premier ordre

- Trouver l'UPG de $p(x, f(x), y)$ et $p(y, z, u)$

- **Itération k=1**
  1. $\sigma_1 = \varepsilon = \{\}$ (σ_k est la valeur courante de l'UPG que l'on construit)

  2. σ_1 unifie-t-elle $p(x, f(x), y)$ et $p(y, z, u)$
     - » **non** : $p(x, f(x), y)\ \sigma_1 \rightarrow p(x, f(x), y) \neq p(y, z, u) \leftarrow p(y, z, u)\ \sigma_1$
     - » alors cherche ensemble de désaccord $D_1 = \{x, y\}$

  3. Existe-t-il un substitution qui unifie les éléments de D_1
     - » **oui** : $\{x = y\}$ (on aurait aussi pu choisir $\{y = x\}$ à la place)
     - » alors met à jour UPG : $\sigma_2 = \sigma_1 \{x = y\} = \{x = y\}$

La logique du premier ordre

- Trouver l'UPG de $p(x, f(x), y)$ et $p(y, z, u)$

- **Itération k=2**
  1. $\sigma_2 = \{x = y\}$

  2. σ_2 unifie-t-elle $p(x, f(x), y)$ et $p(y, z, u)$
     - » **non** : $p(x, f(x), y)\ \sigma_2 \rightarrow p(y, f(y), y) \neq p(y, z, u) \leftarrow p(y, z, u)\ \sigma_2$
     - » alors cherche ensemble de désaccord $D_2 = \{f(y), z\}$

  3. Existe-t-il un substitution qui unifie les éléments de D_2
     - » **oui** : $\{z = f(y)\}$
     - » alors met à jour UPG : $\sigma_3 = \sigma_2\{z = f(y)\} = \{x = y, z = f(y)\}$

La logique du premier ordre

- Trouver l'UPG de $p(x, f(x), y)$ et $p(y, z, u)$

- **Itération k=3**
  1. $\sigma_3 = \{x = y, z = f(y)\}$

  2. σ_3 unifie-t-elle $p(x, f(x), y)$ et $p(y, z, u)$
     - » **non** : $p(x, f(x), y)\ \sigma_3 \rightarrow p(y, f(y), y) \neq p(y, f(y), u) \leftarrow p(y, z, u)\ \sigma_3$
     - » alors cherche ensemble de désaccord $D_3 = \{y, u\}$

  3. Existe-t-il un substitution qui unifie les éléments de D_3
     - » **oui** : $\{y = u\}$ (on aurait aussi pu choisir $\{u = y\}$ à la place)
     - » alors met à jour UPG : $\sigma_4 = \sigma_3\{y = u\} = \{x = u, z = f(u), y = u\}$

- Trouver l'UPG de $p(x, f(x), y)$ et $p(y, z, u)$

- **Itération k=4**
  1. $\sigma_4 = \{x = u, z = f(u), y = u\}$

  2. σ_4 unifie-t-elle $p(x, f(x), y)$ et $p(y, z, u)$
     - » **oui** : $p(x, f(x), y)\ \sigma_4 \rightarrow p(u, f(u), u) = p(u, f(u), u) \leftarrow p(y, z, u)\ \sigma_4$
     - » alors on retourne l'UPG σ_4

La logique du premier ordre

- La résolution est une règle d'inférence importante qui peut être appliquée à une certaine classe de fbf appelées *clauses*
- Une clause est une fbf formée d'une disjonction de littéraux.
- Le processus de résolution est appliqué à deux clauses parentes pour produire une clause dérivée. Donc ce principe de résolution ne peut être appliqué qu'aux clauses.
- Heureusement toute fbf peut être transformée en un ensemble de clauses. Cette transformation est faite à l'aide du processus suivant.

La logique du premier ordre

Transformation d'une fbf en clauses.

Cette transformation comprend les étapes suivantes:

1) Eliminer les implications à l'aide de la règle suivante: $X1 \Rightarrow X2 \equiv \neg X1 \vee X2$

2) Réduire les portées des négations jusqu'aux littéraux avec les lois de De Morgan.

3) Standardiser les variables: Dans la protée de n'importe quel quantificateur, une variable liée par ce quantificateur peut être remplacée par toute autre variable n'apparaissant pas dans la protée de ce quantificateur. (renommer les variables).

Exemple: *$(\forall x)P(x) \Rightarrow (\exists x)Q(x)$ est équivalente à $(\forall x)P(x) \Rightarrow (\exists y)Q(y)$*

La logique du premier ordre

4) Eliminer les quantificateurs existentiels par le processus suivant: Remplacer une variable existentielle par une fonction de Skolem (un nouveau nom de fonction) dont les arguments sont les variables liées à des quantificateurs universels dont la portée inclut la portée du quantificateur existentiel à éliminer. S'il n'existe pas de quantificateur universels alors la fonction de Skolem est une constante de Skolem.

Exemples :

(∀ y)(∃ x) P(x,y) devient (∀ y)P(g(y),y) où g est une fonction de Skolem.

(∀x)(∀y)(∃z) P(x,y,z) devient (∀x)(∀y) P(x,y,g(x,y)) où g est une fonction de Skolem

(∃ x)P(x) devient P(A) où A est une constante de Skolem

La logique du premier ordre

5) Mettre l'expression sous forme normale prenexe. Une fbf sous forme normale prenexe est de la forme:

Préfixe	Matrice
Les quantificateurs ∀	Formule sans les quantificateurs

6) Mettre la matrice sous forme normale conjonctive. On utilise la règle de distributivité suivante: $X1 \vee (X2 \wedge X3) \equiv (X1 \vee X2) \wedge (X1 \vee X3)$

7) Eliminer les quantificateurs universels (effacer la partie préfixe)

8) Eliminer les symboles ∧ en remplaçant $X1 \wedge X2 \wedge X3 \ldots \wedge Xn$ par l'ensemble de clauses {X1, X2, X3, ..., Xn} Chaque Xi est formée de disjonction de littéraux.

9) Renommer les variables des clauses Xi

La logique du premier ordre

- **Résolution dans le cas général :** Pour pouvoir appliquer la résolution à 2 clauses (appelées clauses parentes), il faut trouver une substitution qui puisse être appliquée aux 2 clauses de telles sorte qu'elles contiennent des littéraux complémentaires.
- Pour faciliter la compréhension, représentons une clause (disjonction de littéraux) sous forme d'ensemble de littéraux.

La logique du premier ordre

- Soient C1 et C2 les 2 ensembles représentant les 2 clauses parentes.
- Soient l1 et l2 les 2 littéraux appartenant respectivement aux clauses C1 et C2 de telles sorte qu'elle existe une substitution σ pour que σ(l1) et σ(l2) soient complémentaires.
- Nous disons alors que les deux clauses parentes C1 et C2 se résolvent en la résolvante σ({C1-l1} ∨ σ({C2-l2}). (On suppose qu'on a renommé les variables des deux clauses parentes). (C1-l1 signifie : enlever l1 de C1)

La logique du premier ordre

- à l'inverse de la logique propositionnelle, la logique du premier ordre n'est pas décidable : il n'existe pas d'algorithmes qui retournent oui si une formule est valide, et non si elle ne l'est pas.

Représentation des connaissances

Représentations graphiques

Réseaux sémantiques

Le réseau sémantique a été conçus à l'origine en linguistique pour devenir ensuite un langage pour la représentation de concepts très divers à l'instar de la représentation d'une structure informatique utilisée en IA.

un réseau sémantique est un graphe composé :

- **D'un ensemble de nœuds étiquetés :** représentant généralement des objets,
- **D'un ensemble de liens orientés et étiquetés entre ces nœuds :** représentant généralement des relations entre des objets,
- **D'un ensemble d'opérations d'exploitation de ce graphe :** constituants les mécanismes de raisonnement

Réseaux sémantiques

La représentation graphique

un nœud est généralement représenté au moyen d'un rectangle, d'un cercle, d'une ellipse.

Un arc relie un nœud source (queue de l'arc) à un nœud cible (tête de l'arc).

La représentation graphique facilite la lecture, ne correspond généralement pas au formalisme d'implémentation,

représentation non-graphique:

(Ali, lire, livre)

Réseaux sémantiques

Les éléments de base:

les NŒUDS

- atomiques : entités élémentaires (valeurs, individus,...)
- complexes : entités complexes (propositions, phrases,...)

ils doivent être typés : concept, individu, action, proposition, etc...

les LIENS

- structuraux: indépendants de la sémantique du domaine,
- spécifiques: dépendants de la sémantique du domaine,
- il faut essayer d'augmenter la proportion des liens structuraux par rapport aux liens spécifiques

Réseaux sémantiques

- **les OPERATIONS**
- souvent représentées par le programme,
- doivent être définies clairement,

Les réseaux sémantiques utilisent généralement deux relations très particulières concernant des objets de l'un des types individu ou classe :

est_un : relation entre un individu et une classe exprimant l'**appartenance** ;

sorte_de : relation entre deux classes exprimant l'**inclusion**.

Réseaux sémantiques

lien structurel indépendant du domaine représente une inclusion

- de propriétés (point de vue intentionnel, cas général)
- d'individus (point de vue extensionnel)

Réseaux sémantiques

"les canaris /sont des /oiseaux"

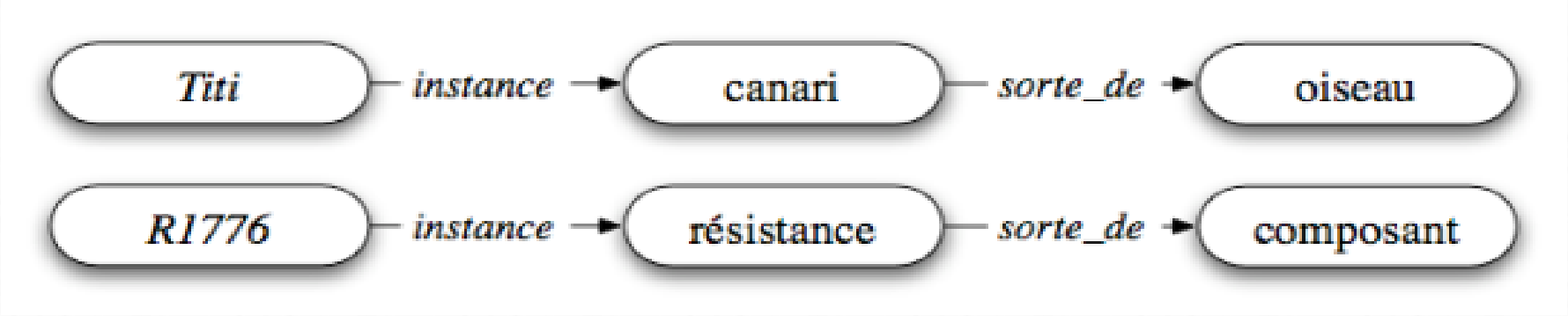
canaris et oiseaux = concepts (nom communs) --> classe
sont des = relation --> inclusion de classes

lien « sorte_de »

Réseaux sémantiques

"Titi /est un /canari"
canari = concepts
Titi = individu (nom propre) --> élément d'un ensemble
est_un = relation --> appartenance d'un élément à une classe

lien « instance »



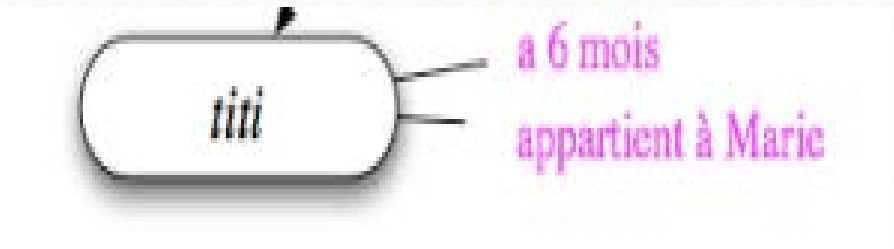
lien « instance » = lien structurel

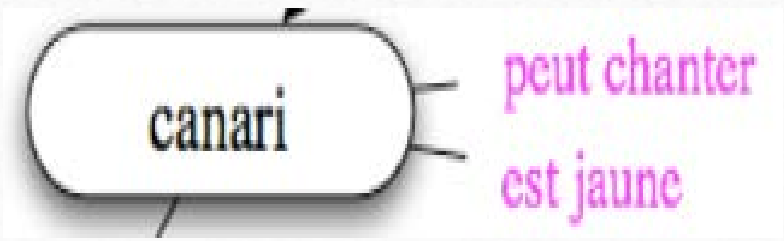
Réseaux sémantiques

Les propriétés sont des informations rattachées à chaque nœud du réseau sémantique :

- Elles sont Simples
- Elles ne permettent pas de répondre à des questions comme :

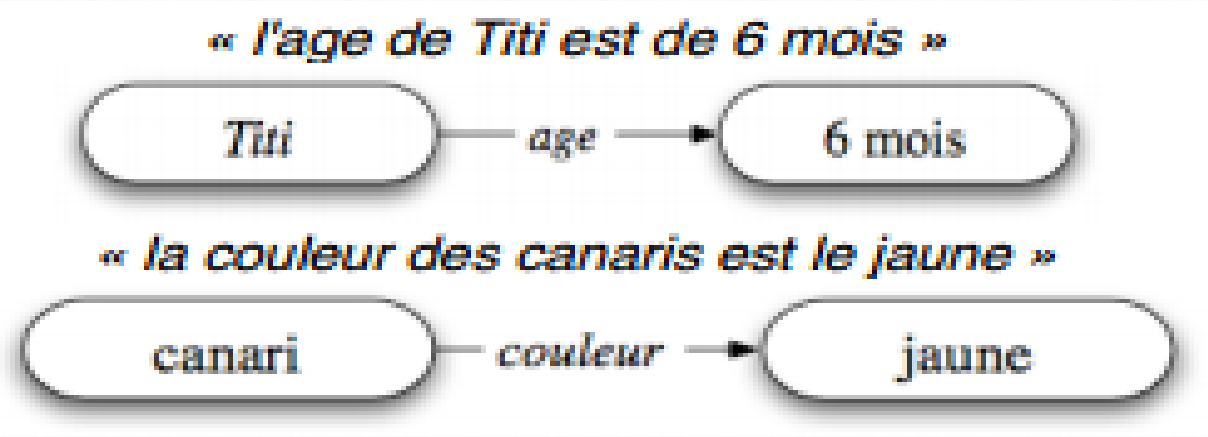
"quel est l'âges de Titi ?" "quelle est la couleur des canaris ? "





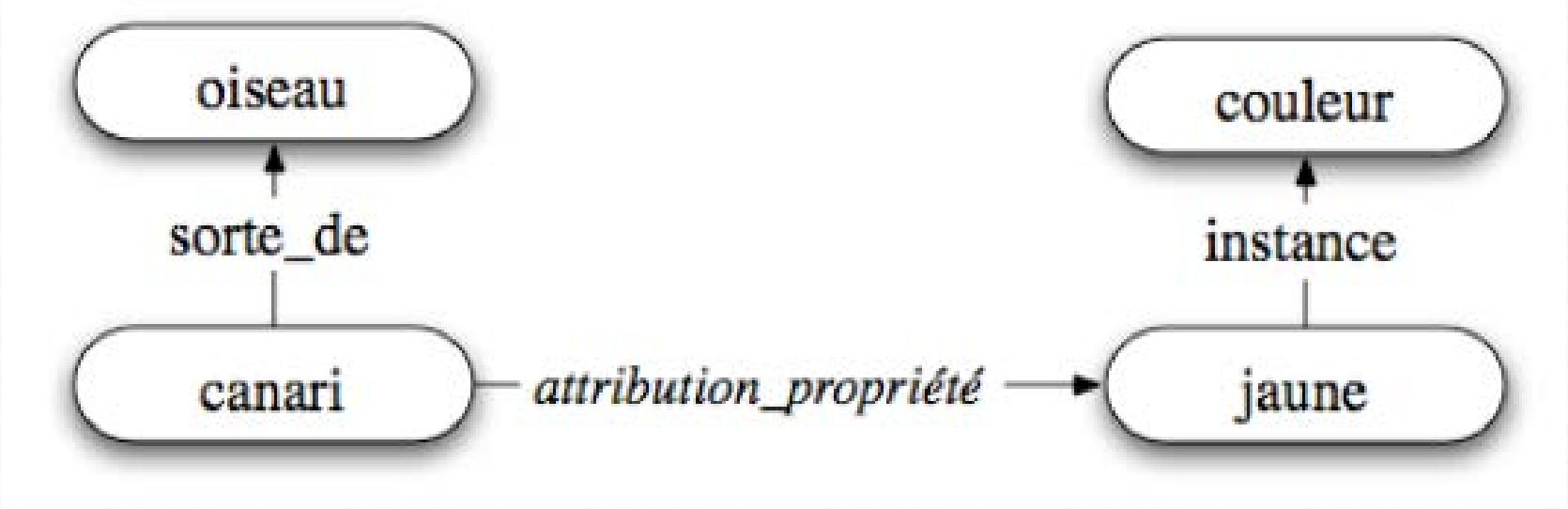
Réseaux sémantiques

Un attribut est une relation qui relie un nœud concept ou un nœud individu à une valeur ou propriété.



Réseaux sémantiques

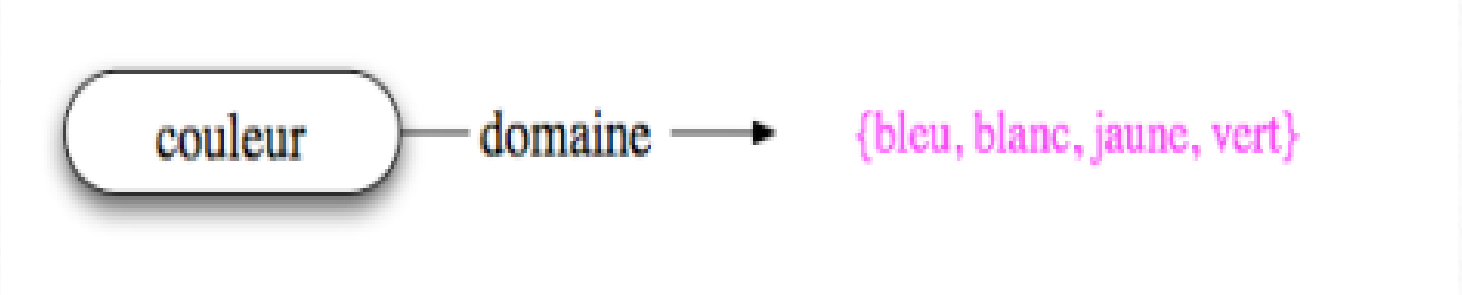
lien spécifique dont le sens dépend du domaine d'application -> interprétation ad-hoc, on peut le rendre plus structurel en créant un nœud-attribut:



Réseaux sémantiques

Un attribut est une classe sémantique de nœud dont les instances sont des propriétés.

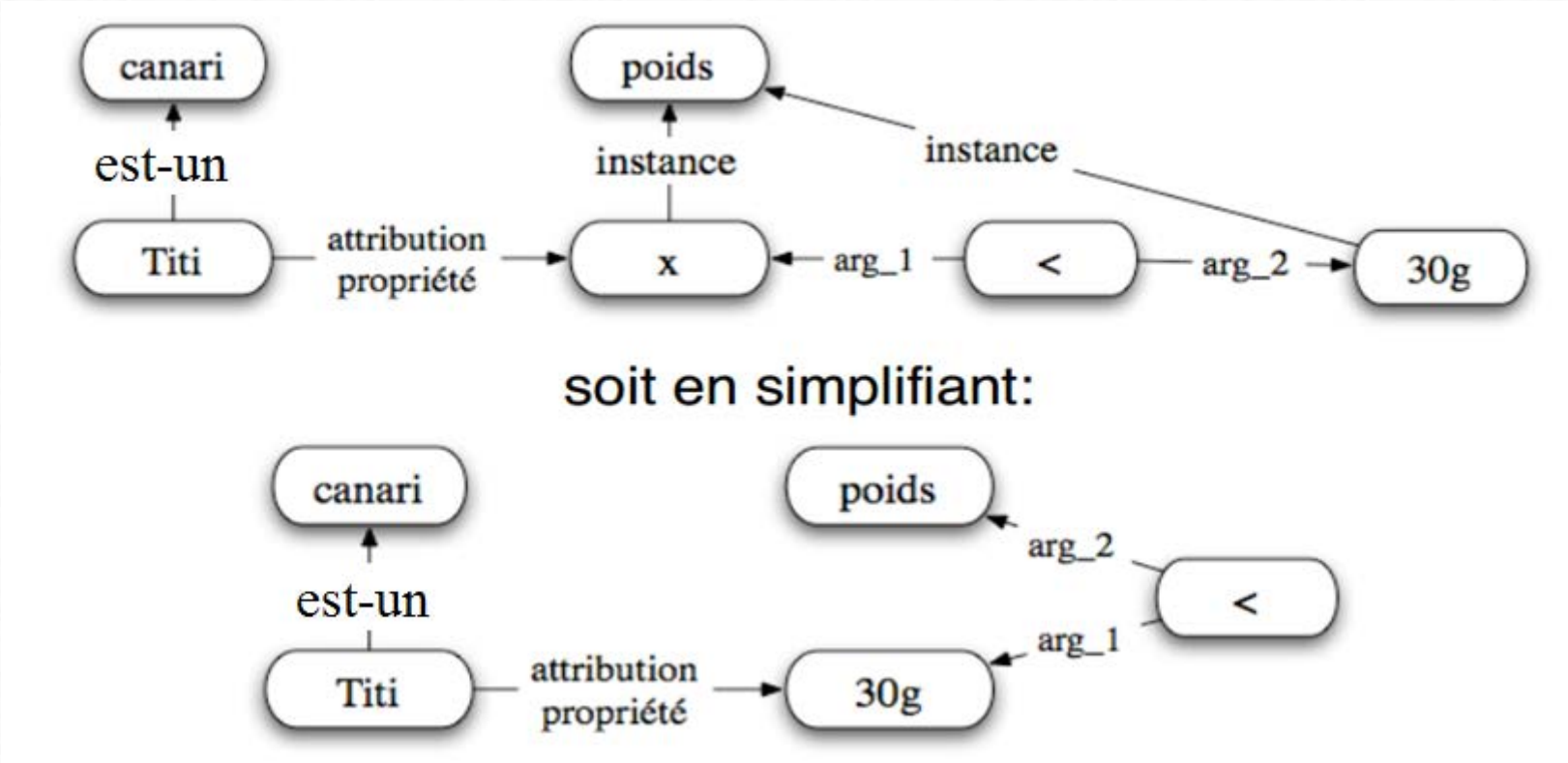
un attribut peut lui-même être caractérisé :



Le domaine est une relation structurelle permettant de vérifier des contraintes d'intégrités.

Réseaux sémantiques

un nœud-attribut peut être relié à une ou plusieurs valeurs par l'intermédiaire d'un opérateur:



Réseaux sémantiques

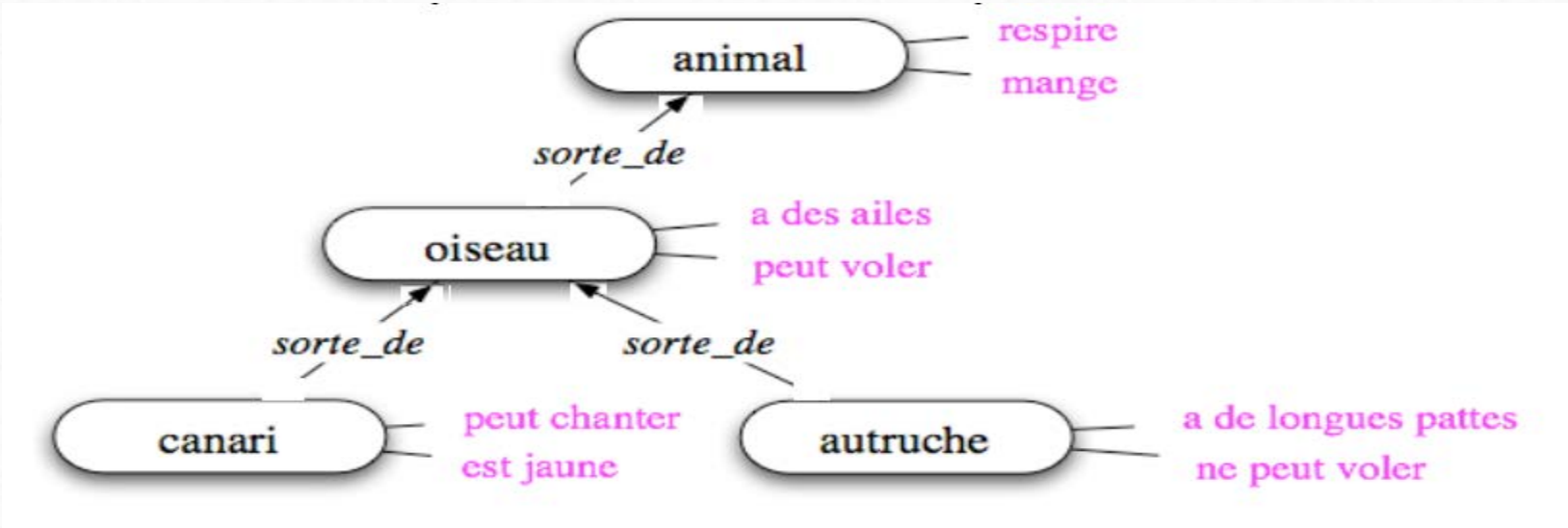
L'héritage dans les réseaux sémantiques repose sur des liens de type « est_un» ou «sorte_de » reliant un concept à un autre concept plus élevé :
exemple: "canari" est une sorte de "oiseau"

héritage des propriétés rattachées au concept père et au concept fils :
Ainsi, on pourra dire que « le canari a des ailes et une peau » en remontant les liens « sorte_de »

Réseaux sémantiques

le principe d'héritage permet :

- de nombreuses déductions automatiques
- de définir la notion de distance sémantique entre deux concepts (nombre de liens devant être traversés pour aller d'un concept à l'autre).



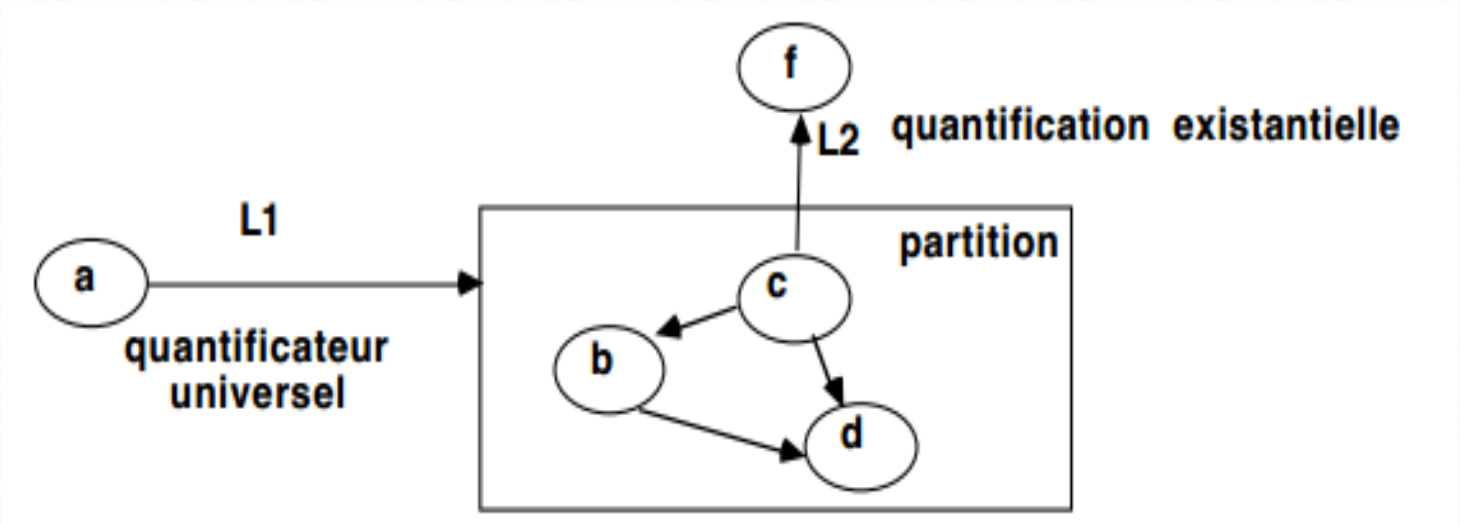
Réseaux sémantiques

La partition est le regroupement de nœuds et d'arcs du réseau dans des espaces spécifiant la portée de relations.

intérêts des partitions :

- définition de contextes
- permet la quantification

Réseaux sémantiques



cadres : définissent l'étendue des identificateurs universels

lien L1 : quantification universelle, quelque soit a, pointe sur un cadre représentant l'étendue de la variable quantifiée universellement.

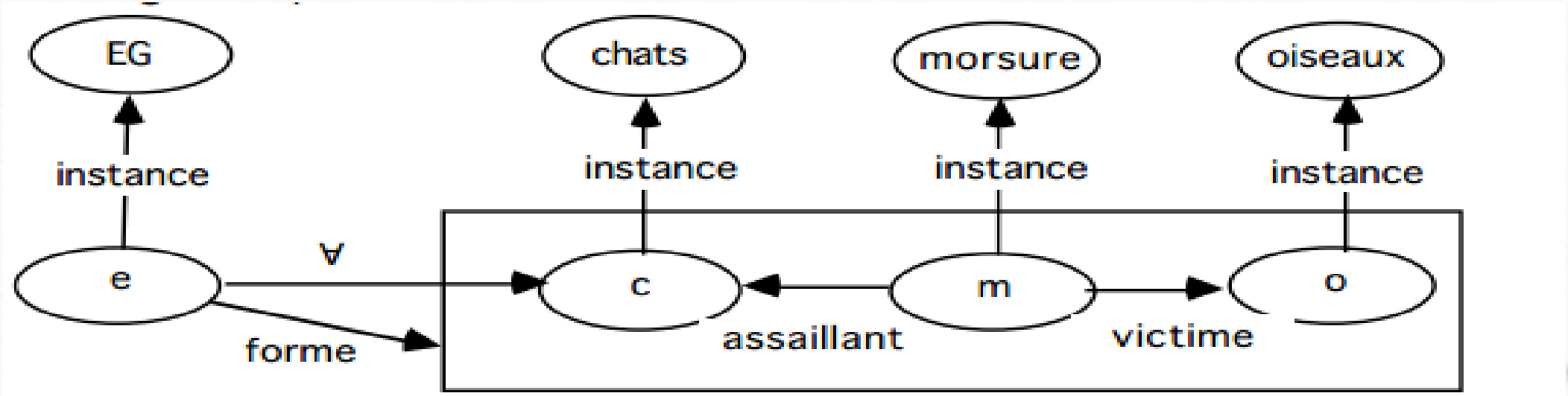
lien L2 : quantificateur existentiel explicite sur le nœud f par rapport au nœud c

Réseaux sémantiques

Le quantification est traitée par la notion de partition

- Soit le fait à représenter suivant : « tout chat a mordu un oiseau »
- représentation logique : "∀ x chat (x) ⇒(∃ y oiseau (y) ∧ mordre (x,y))"

encodage de la variable quantifiée universellement x en utilisant une partition (cadre rectangulaire) :



Réseaux sémantiques

- les nœuds **c, m, o** sont des instances de chats, morsure, oiseaux,
- le cadre introduit dans le réseau définit l'étendue de l'identificateur universel,
- le nœud **e** représente l'assertion à représenter, instance de l'ensemble des énoncés généraux EG sur le monde,
- chaque élément de EG possède :
  - une connexion « forme » pointant vers le cadre de la partition et énonce l'affirmation,
  - une ou plusieurs connections « ∀» pointant vers chaque variable quantifiée universellement, ici variable c (les variables m et o sont ici quantifiée existentiellement).

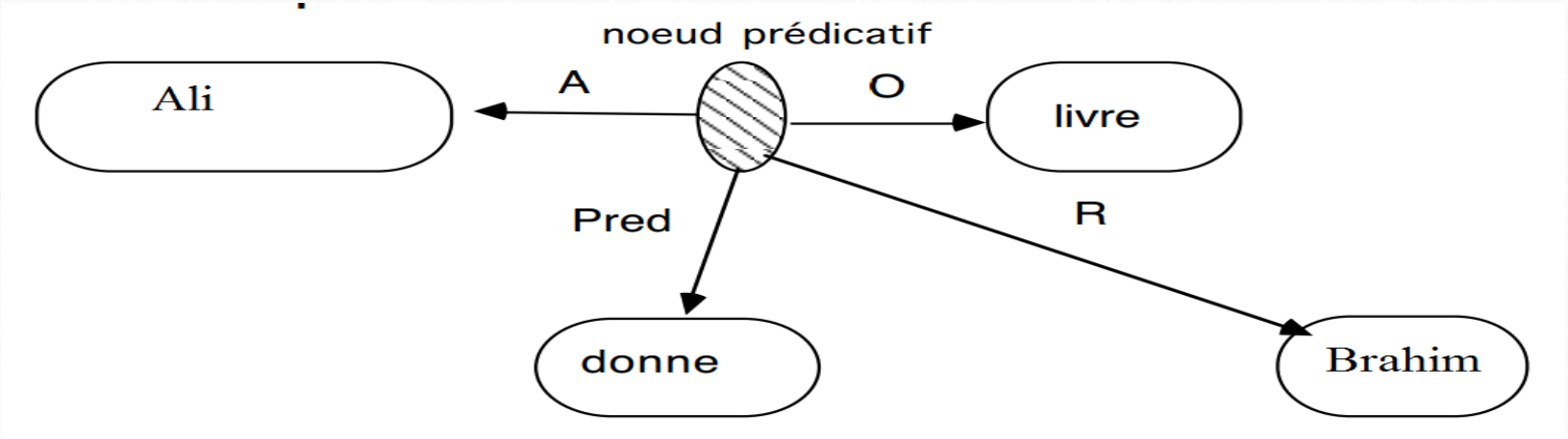
Réseaux sémantiques

Travaux de Schubert et Cerone :

- introduction de la logique des prédicats du premier ordre
- introduction d'un "nœud prédicatif" instancié en lui associant :
  - un pointeur vers le prédicat
  - un pointeur vers chaque argument du prédicat

Réseaux sémantiques

Exemple: soit la phrase Ali donne un livre à Brahim

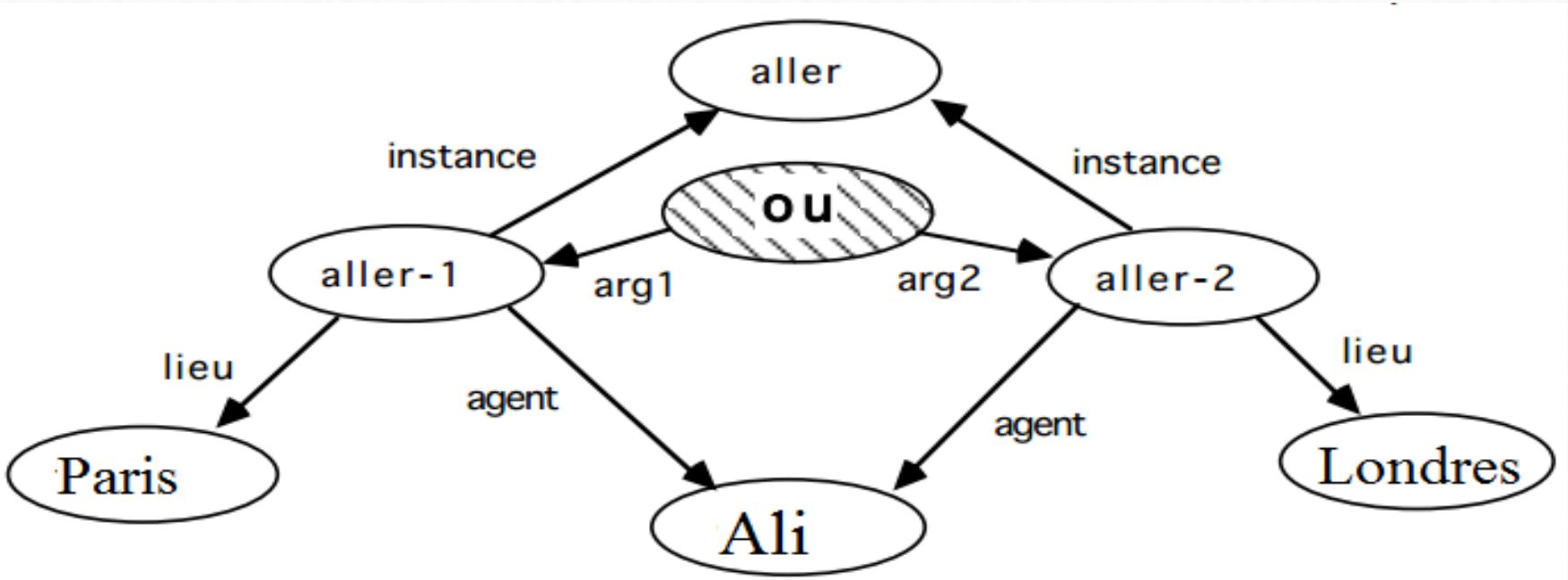


soit : donne (A, O, R)
avec : A (agent) = Ali; O (objet) = livre; R (receveur) = Brahim

Réseaux sémantiques

Les connecteurs logiques ET, OU

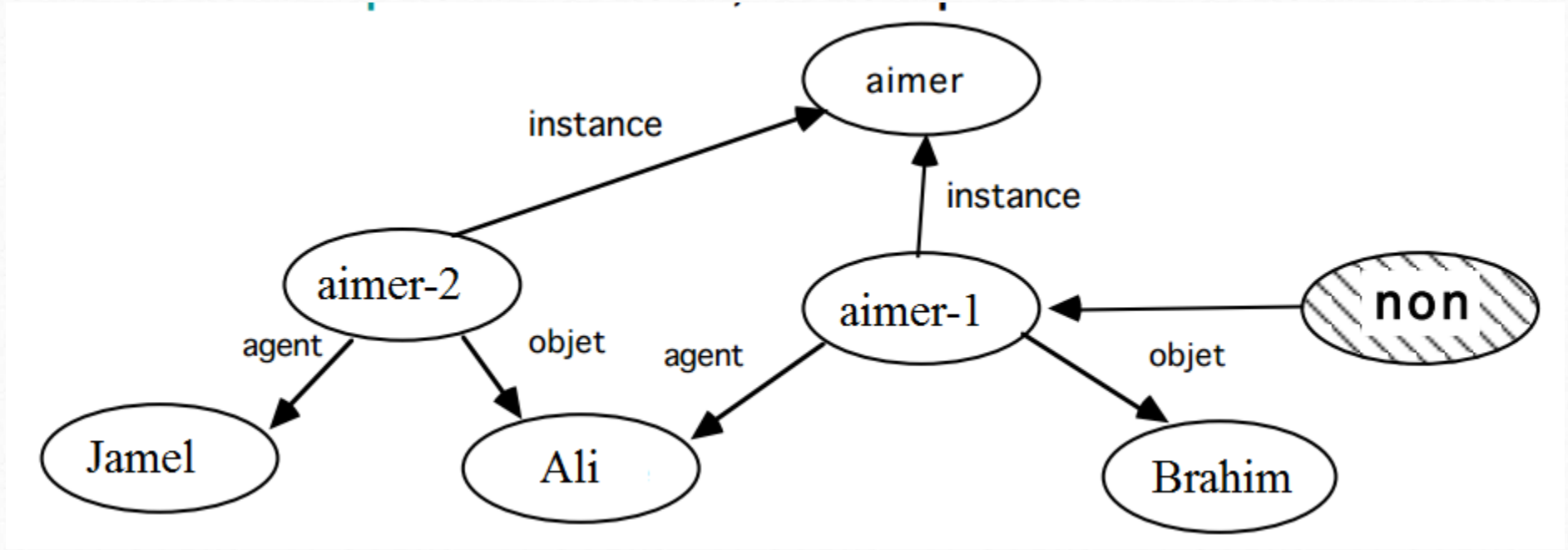
Soit la phrase suivante: « Ali ira à Paris ou à Londres», elle est représenté comme ceci :



Réseaux sémantiques

Représentation de la négation

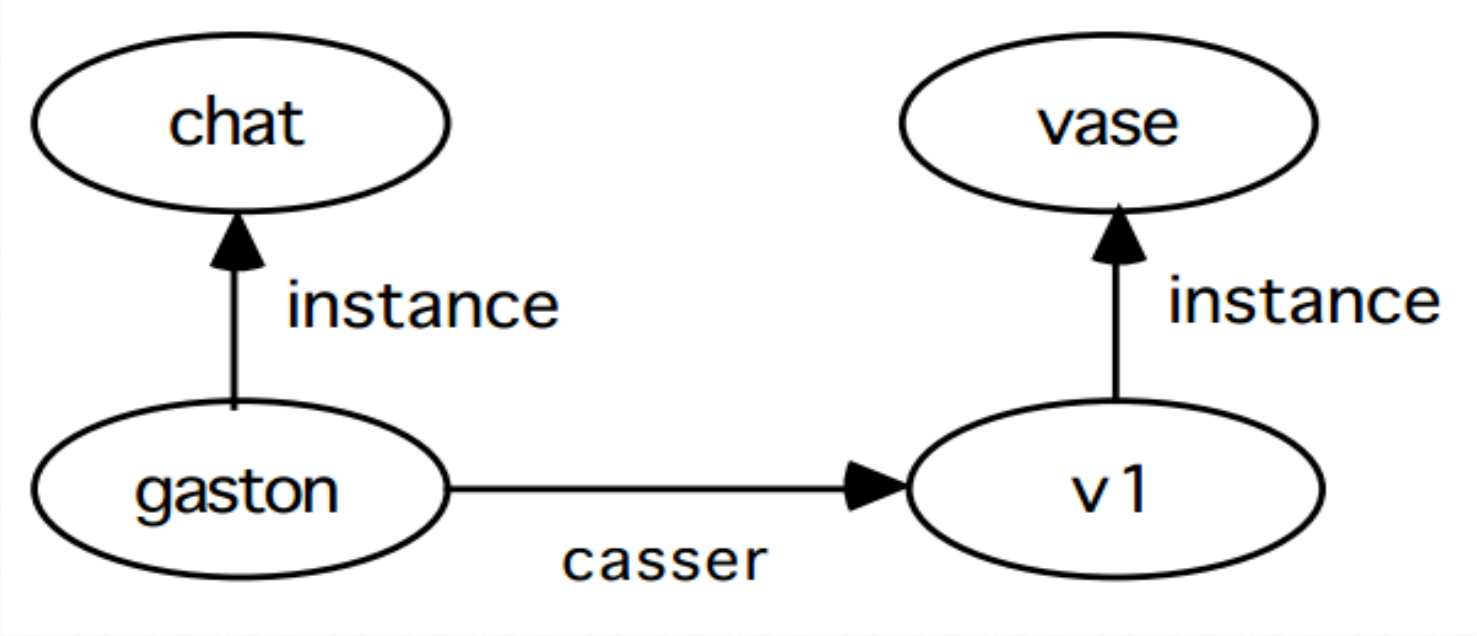
Soit la phrase: « Ali n'aime pas Brahim », la représentation est :



Réseaux sémantiques

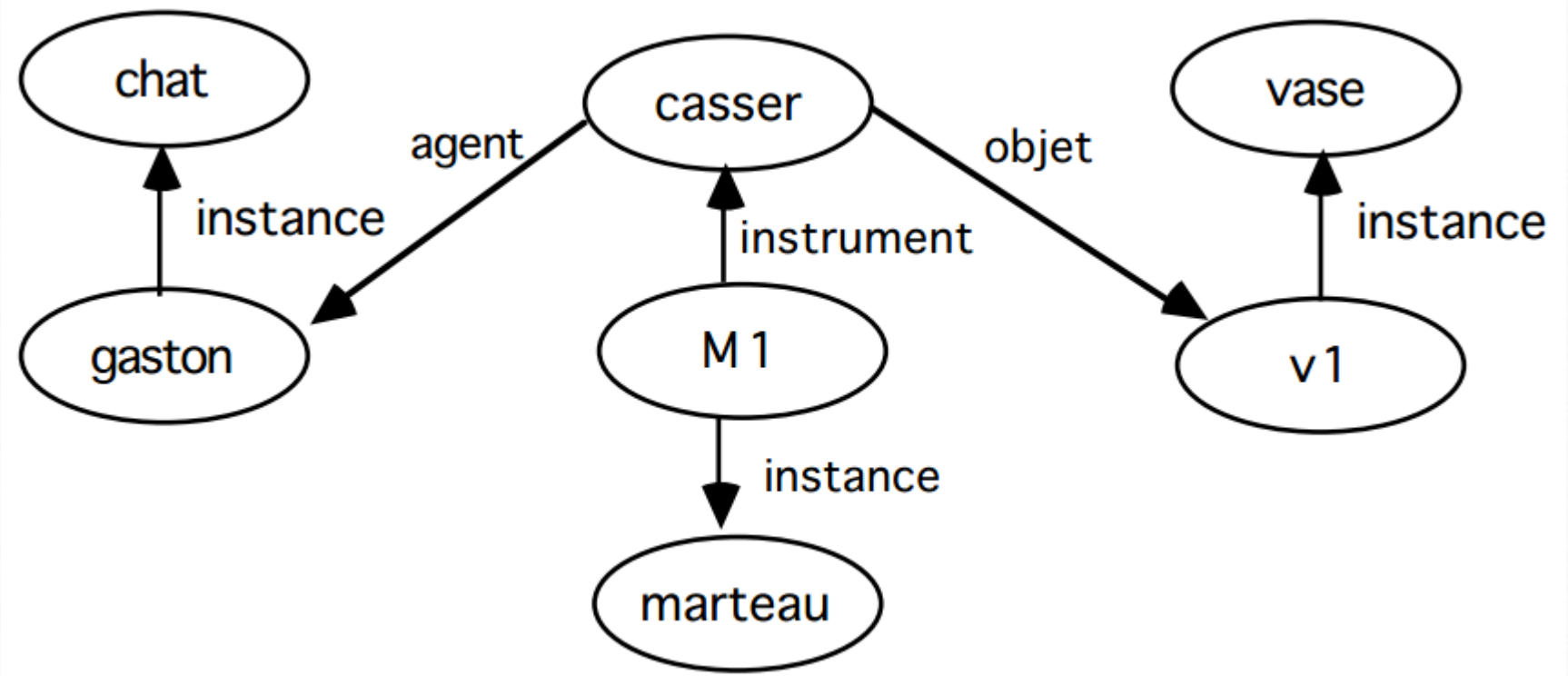
Représentation d'évènements ou d'actions

Représenter l'événement : « Gaston casse le vase » :



Réseaux sémantiques

le lien « casser » est spécifique. On peut s'en séparer en le traduisant par des liens plus structurels : agent, objet, instrument, temps, lieu, ... :



Réseaux sémantiques

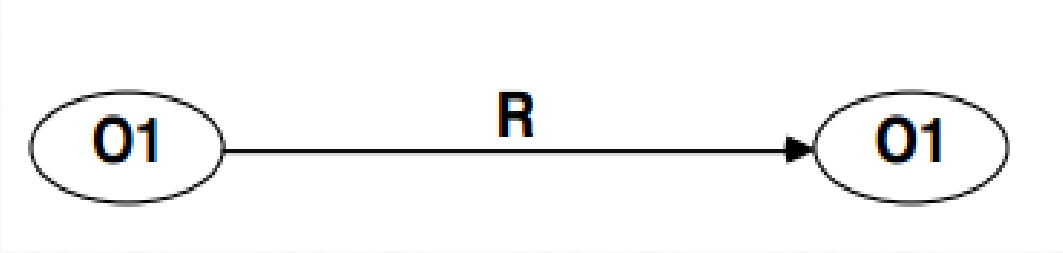
Interprétation des connaissances dans les réseaux sémantiques

- l'accès aux données stockées dans un **réseau sémantique** n'est pas assuré par le réseau lui-même
- On fait appel à un interpréteur (transformer les données du réseau en connaissances opératoires)
- on doit disposer alors de:
  - soit d'un langage élaboré de navigation et d'inférence dans le réseau,
  - soit d'un langage limité à l'accès dans le réseau + autre programme assurant les inférences

Réseaux sémantiques

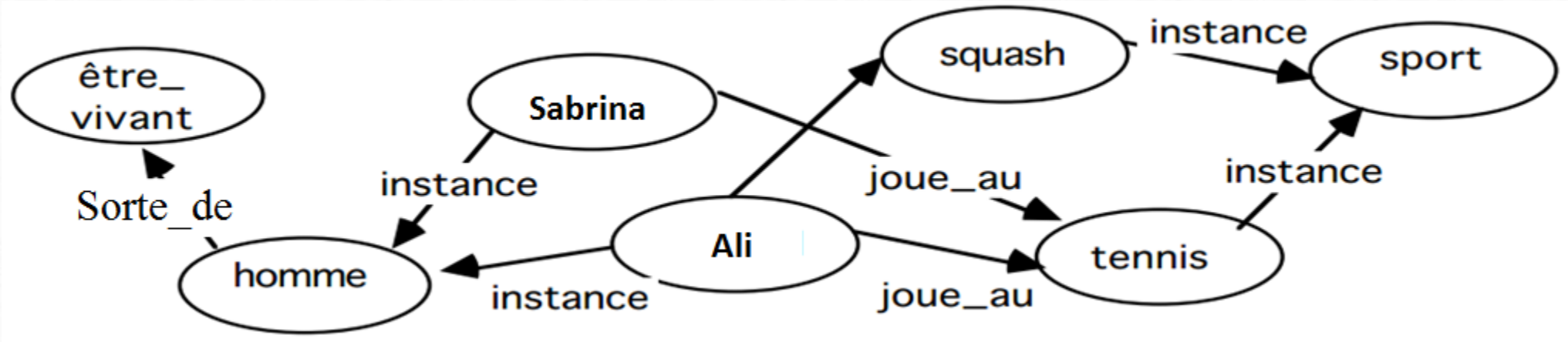
Exemple interprétation d'un réseau sémantique par règles de production (Snark) :

- moteur d'inférences à règles de production à variables
- logique d'ordre 0,1,2
- faits = triplets (O1, R, O2)

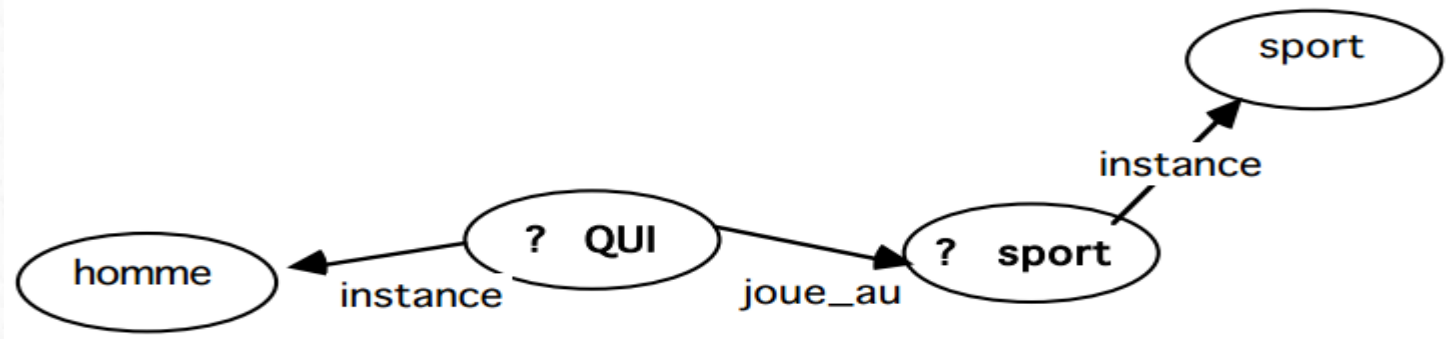


Réseaux sémantiques

On dispose d'une base de connaissances organisée en réseau sémantique :



la question « quelqu'un fait-il du sport ? » représentée par le fragment de réseau suivant :



Réseaux sémantiques

les réponses seront :

- sabrina joue_au tennis
- Ali joue_au tennis
- Ali joue_au squash

si le réseau est important donc on aura des problèmes combinatoires

Réseaux sémantiques

Intérêts des réseaux sémantiques

Le réseau sémantique offre des axes organisationnels pour structurer une base de connaissances :

La classification, ensembles/sur-ensembles

Un objet peut être associé avec son ou ses types génériques,

Ex: Titi peut être associé à oiseau, animal.

- conduit à la distinction fondamentale de type (canari) et d'occurrence (Titi).
- peut être récursive -> définir des méta-types ayant pour instance d'autres types.

Réseaux sémantiques

Agrégation

- rattacher à un objet des propriétés ou d'autres objets y intervenant comme parties.

Ex: Titi, vu comme objet physique possède des ailes, une tête et une queue, considéré dans son environnement, il possède un nid, un territoire, un chant, une nourriture.

- peut être appliqué récursivement: un composant peut être à son tour composé d'autres composants

Réseaux sémantiques

La généralisation, la spécialisation

relie un type à un autre type plus générique,
Ex: oiseau à animal

- La généralisation (lien « sorte_de ») = un ordre partiel organisant deux types dans une généralisation ou une hiérarchie.
- économie de place en mémoire (propriétés associées à des types généraux hérités par d'autres types plus spécialisés).
- généralisation plus facile de grandes bases de connaissances (bases de données)

Réseaux sémantiques

La partition

regroupe des objets et éléments de relations dans des partitions qui sont organisées de façon hiérarchiques;

Ex: si une partition P1 est au-dessous d'une autre P2, toute chose visible ou présente dans P2 l'est aussi dans P1, sans pour autant l'y avoir été spécifiée.

- principal intérêt = quantification, la représentation du temps et de l'hypothétique.

Réseaux sémantiques

Forces des RS :

- les objectifs d'extraction de connaissance dans la base de connaissances s'expriment en chemins de traversée sur la structure même de la base
- possèdent des principes d'organisation relativement puissants (généralisation, partition, agrégation) permettant de structurer la base de connaissances
- formalisme graphique : bonne compréhension, intéressant à un premier stade de formalisation de la connaissance
- formalisation déclarative : finesse et cohérence de représentation des concepts

Réseaux sémantiques

Faiblesses des RS :

- manque de sémantique formelle et de terminologie standard
- interprétation difficile des connaissances : toujours un compromis à faire entre la complexité d'une structure de données et la complexité de l'interpréteur
- critique si taille du réseau importante (nb de nœuds et liens) ! explosion combinatoire

Résolution des problèmes

Méthodes de recherche et de contrôle

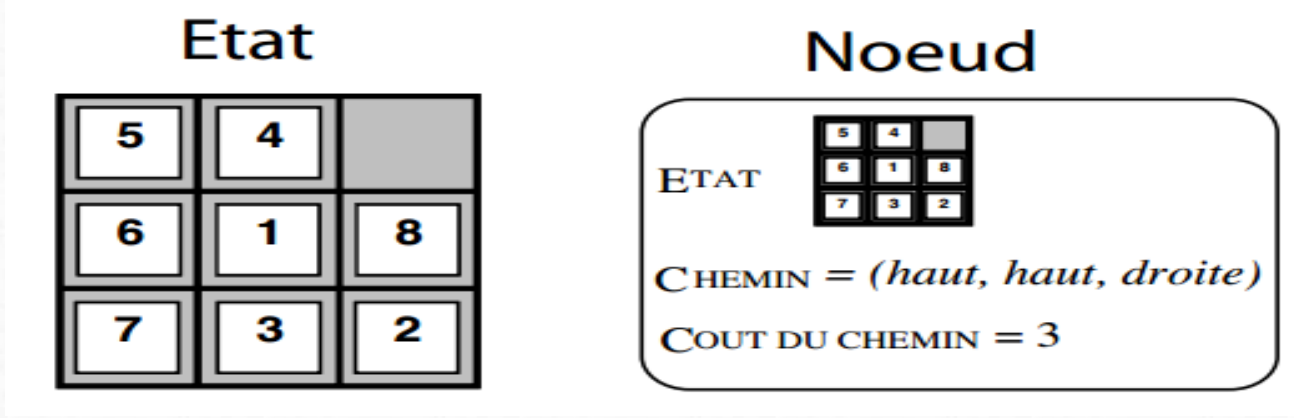
Méthodes de recherche et de contrôle

Il existe plusieurs algorithmes spécifiques, avant de détailler ces algorithme spécifiques nous présentons un algorithme de recherche générique suivant:

fonction recherche(*état_initial*, successeurs, test_but, coût)
nœuds_à_traiter ← créer_liste(créer_nœud(*état_initial*, [], 0))
boucle
si vide ?(*nœuds_à_traiter*) **alors renvoyer** échec
nœud ← enlever_premier_noeud(*nœuds_à_traiter*)
si test_but(état(*nœud*)) = *vrai*
alors renvoyer chemin(*nœud*), état(*nœud*)
pour tout *(action, état)* **dans** successeurs(état(*nœud*))
chemin ←[*action*, chemin(*nœud*)]
coût_du_chemin ← coût_du_chemin(*nœud*) + coût (*état*)
s ← *créer_nœud*(*état*, *chemin*, *coût_du_chemin*)
insérer(*s*, *nœuds_à_traiter*)

Méthodes de recherche et de contrôle

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 de nœuds à traiter, un nœud étant compose d'un état, d'un chemin, et du coût du chemin.



Nous commençons avec un seul nœud correspondant à l' état initial.

Méthodes de recherche et de contrôle

A chaque itération de la boucle, nous vérifions si la liste de nœuds à traiter est vide. Si c'est le cas, nous avons examine 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.

Méthodes de recherche et de contrôle

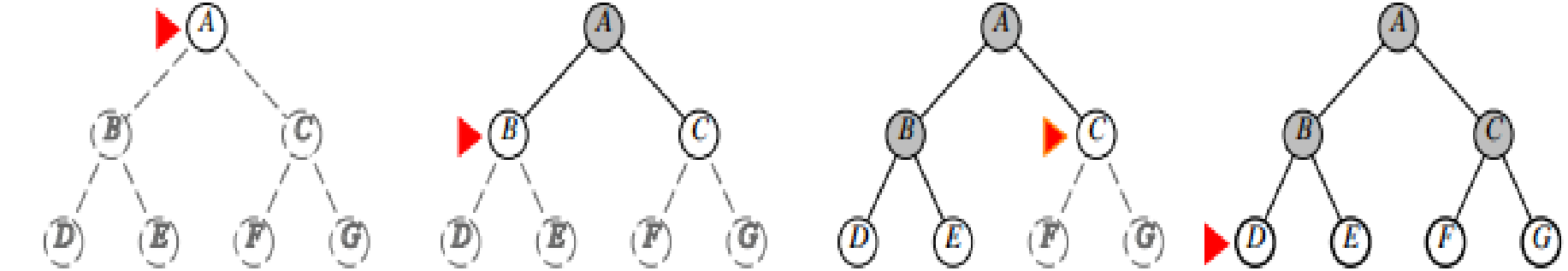
Recherche non-informée

Les algorithmes de recherche *noninformés sont d*es algorithmes 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.

Méthodes de recherche et de contrôle

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 suivante.



Méthodes de recherche et de contrôle

Le parcours en largeur est un algorithme de recherche complet 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.

Méthodes de recherche et de contrôle

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.

Méthodes de recherche et de contrôle

Considérons ensuite la complexité de cet algorithme. Supposons que chaque état possède *s* successeurs et que *p* soit le nombre minimal d'actions pour relier l'état initial à un état but.

Dans le pire des cas, nous allons examiner tous les nœuds de profondeur au plus *p* afin de trouver l'état but qui se trouve à cette profondeur. Nous allons alors produire: $1 + s + s^2 + s^3 + ... + s^p + (s^{p+1} - s)$ nœuds.

Méthodes de recherche et de contrôle

La quantité $s^{p+1} - s$ dans cette formule correspond au nombre de nœuds de profondeur $p + 1$ qui ont été produits lors de l'examen des nœuds de profondeur p moins les successeurs du dernier nœud (comme le dernier nœud est le bon, nous ne produisons pas ses successeurs). Le parcours en largeur a donc une complexité en temps de $O(s^{p+1})$. La complexité en espace est aussi en $O(s^{p+1})$ puisque nous avons au plus $1 + s^{p+1} - s$ nœuds en mémoire lors de l'examen du dernier nœud.

Méthodes de recherche et de contrôle

La figure suivante montre le temps et la mémoire nécessaires au parcours en largeur pour un problème ayant 10 successeurs par état ($s = 10$). Nous voyons que même pour $p = 6$, le parcours en largeur utilise 10 giga-octets de mémoire, et pour $p = 8$, il lui faut 1 téra-octet ! Le temps de calcul pose aussi problème : l'algorithme prend 35 ans pour trouver une solution de profondeur $p = 12$. Cette haute complexité en temps et en espace restreint l'utilisation du parcours en largeur aux petits problèmes.

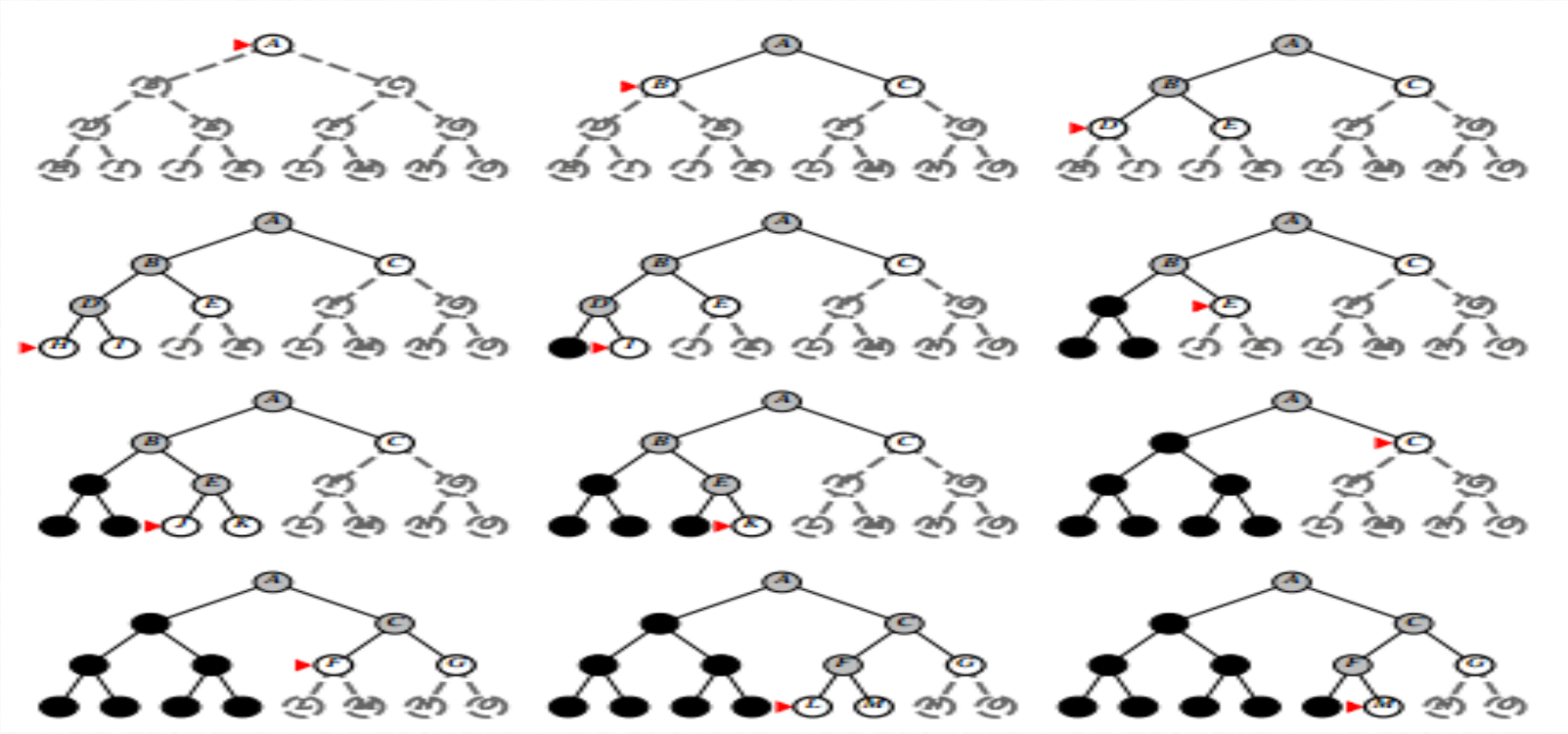
Profondeur	Noeuds Générés	Temps		Mémoire	
2	1100	0,11	sec	1	mega-octet
4	111 100	11	sec	106	mega-octet
6	10^7	19	min	10	giga-octet
8	10^9	31	h	1	tera-octet
10	10^{11}	129	jours	101	tera-octets
12	10^{13}	35	ans	10	peta-octets

.

Méthodes de recherche et de contrôle

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 suivante montre le fonctionnement de cet algorithme sur un petit exemple.



Méthodes de recherche et de contrôle

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. Considérons maintenant la complexité. Si nous avons s successeurs et une profondeur maximale de m actions, dans le pire des cas nous aurions besoin d'examiner tous les s^m nœuds pour trouver une solution.

Méthodes de recherche et de contrôle

Sur ces trois premiers critères, il est évident que le parcours en largeur sort gagnant. Alors pourquoi s'intéresser au parcours en profondeur ? Le principal avantage du parcours en profondeur reste en sa faible complexité en espace.

Pour un problème ayant s successeurs et un profondeur maximal de m actions, il nous faut garder au plus $1+(m*(s-1))$ nœuds en mémoire (correspondant au chemin actuellement en développement plus les successeurs de profondeur 1 que nous n'avons pas encore traités, les successeurs de profondeur 2 que nous n'avons pas encore traités, etc.).

Méthodes de recherche et de contrôle

La complexité en espace est donc de $O(s^* m)$. Même quand s et m sont grands, la mémoire nécessitée par le parcours en profondeur reste raisonnable, ce qui nous permet de traiter des problèmes qui ne sont pas abordables par le parcours en largeur.

Méthodes de recherche et de contrôle

Parcours en profondeur limitée

Nous venons de voir que le parcours en profondeur n'est pas très bien adapté aux problèmes où la longueur des chemins n'est pas bornée parce que nous risquons de suivre aveuglement un chemin infini qui ne mène pas à un état but.

Une façon naïve d'éviter ce problème serait de fixer une profondeur maximale et de ne pas considérer les chemins de profondeur supérieur à cette limite. L'algorithme résultant, nommé parcours en profondeur limitée, va toujours terminer (à condition bien sûr que le nombre de successeurs soit fini) ce qui n'était pas le cas pour le parcours en profondeur simple.

Méthodes de recherche et de contrôle

Le parcours en profondeur limitée est complet si la profondeur maximale est supérieure à la profondeur minimale des solutions. Mais comme nous savons pas en général quelle profondeur sera suffisante, nous ne saurons pas si l'algorithme est complet ou non pour une profondeur donnée.

Comme c'était le cas pour le parcours en profondeur classique, le parcours en profondeur limitée n'est pas optimal en général. Pour une profondeur maximale fixée à m et s successeurs par état, la complexité en temps sera s^m (dans les pire des cas il faut examiner chacun des s^m chemins de profondeur au plus m). La complexité en espace est d'ordre $O(s^* m)$.

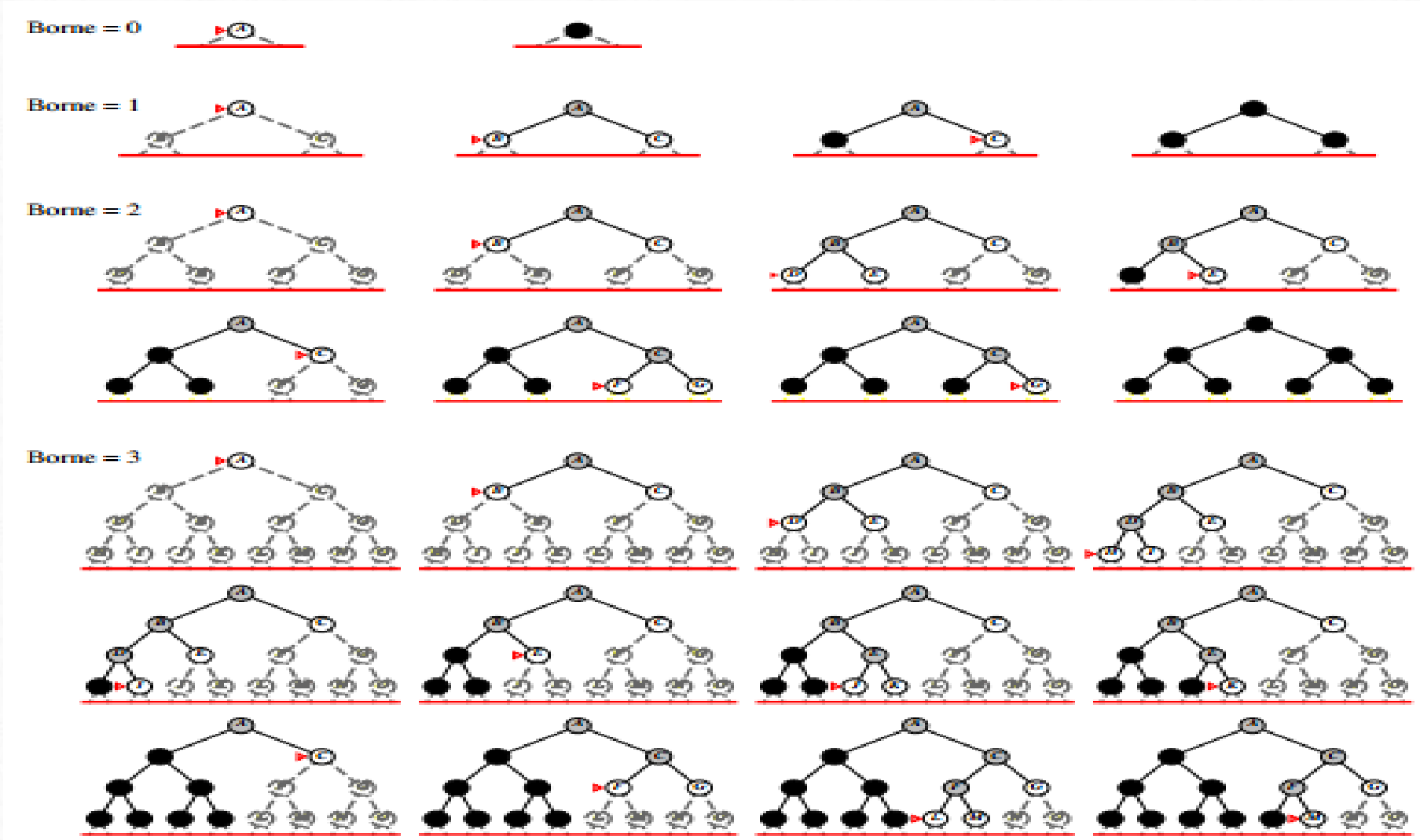
Parcours en profondeur itérée

l'inconvénient principal du parcours en profondeur limitée est la difficulté de bien choisir la borne de profondeur.

Le parcours en profondeur itérée permet de remédier (de façon un peu brutale !) à cet inconvénient : comme nous ne savons pas quelle borne de profondeur choisir, nous allons les essayer les unes après les autres. Nous effectuons donc un parcours en profondeur limitée avec une borne de 1, puis un parcours de profondeur avec une borne de 2, et nous continuons a augmenter la borne jusqu'à ce que l'on trouve une solution.

La Figure suivante montre le déroulement de l'algorithme sur un exemple.

Méthodes de recherche et de contrôle



Méthodes de recherche et de contrôle

Comme le parcours en largeur, le parcours en profondeur itérée est complet — à condition que le nombre de successeurs soit toujours fini — et optimal quand le coût d'un chemin ne dépend que du nombre d'actions.

Pour un problème ayant s successeurs par état et une solution de profondeur p, nous allons examiner $p + 1$ fois le nœud unique de profondeur 0, p fois les nœuds de profondeur 1, $p - 1$ fois les nœuds de profondeur 2, ..., et une seule fois les nœuds de profondeur p.

.

Méthodes de recherche et de contrôle

Le nombre de nœuds génère lors de la recherche est donc:

$(p + 1) + (p)s + (p - 1)s^2 + (p - 2)s^3 + ... + (1)s^p$ ce qui donne une complexité en temps en $O(s^p)$.

Remarquons que même s'il semble inefficace de traiter les mêmes nœuds plusieurs fois, la complexité en temps du parcours en profondeur itérée est en fait moins importante que celle du parcours en largeur (où la complexité est $O(s^{p+1})$ parce que nous produisons des nœuds de profondeur $p+1$ lors de l'examen des nœuds de profondeur p) et que celle du parcours en profondeur classique (où nous pouvons examiner les nœuds de profondeur supérieure à p).

Quant à la complexité en espace, nous gardons au plus $1 + (p * (s - 1))$ nœuds en mémoire lors de l'examen des nœuds de profondeur p, donc la complexité en espace est en $O(s * p)$.

Méthodes de recherche et de contrôle

Le parcours en profondeur itérée combine donc les avantages du parcours en largeur (complétude et optimalité) et du parcours en profondeur (faible complexité en espace).

C'est pour cette raison que le parcours en profondeur itérée est considéré aujourd'hui comme le meilleur algorithme de recherche non-informée pour les problèmes de grande taille où la profondeur des solutions est inconnue.

Méthodes de recherche et de contrôle

Etats redondants

Les algorithmes que nous venons de présenter peuvent visiter plusieurs fois le même état pendant la même recherche.

Il y a deux types de redondances possibles: d'une part, il y a des états qui peuvent être visités deux fois sur le même chemin (par exemple, au taquin, si vous jouez droite puis gauche, vous revenez à l'état initial), et d'autre part, il peut y avoir plusieurs chemins différents qui amènent au même état.

Dans ce qui suit, nous considérons les façons d'éviter de telles redondances.

Méthodes de recherche et de contrôle

Il est relativement simple d'éviter le première type de redondance. Il suffit d'examiner au fur et à mesure les chemins des nœuds à insérer et de ne jamais ajouter à la liste de nœuds à traiter des nœuds dont les chemins contiennent plusieurs fois le même état.

Il est possible de modifier les différents algorithmes de telle façon que cette modification ne change pas significativement la complexité en espace. Par contre, si nous voulons éviter le deuxième type de redondance, cela signifie qu'il faut se rappeler de tous les états déjà visités et de ne garder qu'un seul chemin par état.

Méthodes de recherche et de contrôle

Plus spécifiquement, il faut gérer une liste des états déjà visités et y ajouter les états quand ils sont visités pour la première fois. Avant de générer les successeurs d'un nœud, il faut vérifier si l'état successeur est dans la liste des états déjà visités (et si c'est le cas, nous ne générons pas de successeurs).

Cette modification ne changera pas trop la complexité en espace pour le parcours en largeur (qui souffre déjà d'une haute complexité en espace). Par contre, pour le parcours en profondeur et le parcours en profondeur itérée qui gardent normalement peu de nœuds en mémoire cette modification va augmenter très significativement la complexité en espace.

Résolution des problèmes

Méthodes de recherche et de contrôle

Méthodes de recherche et de contrôle

Recherche Heuristique

Les algorithmes que nous avons vus précédemment 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.

Méthodes de recherche et de contrôle

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

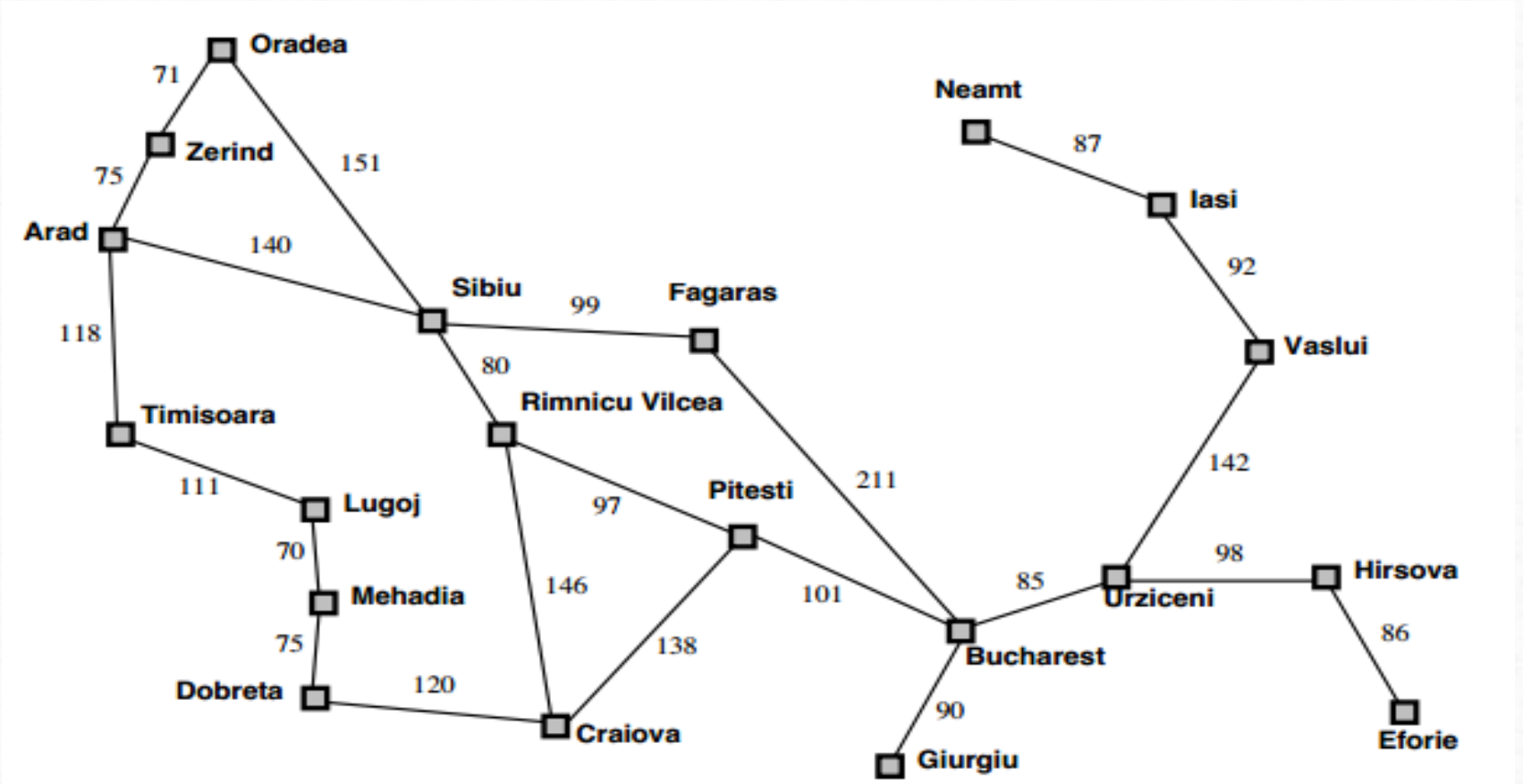
$h(n)$ = coût estimé du chemin de moindre coût reliant n à un état but.

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.

Méthodes de recherche et de contrôle

La figure suivante 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.



.

Méthodes de recherche et de contrôle

Pour mesurer à quel point nous sommes proches du but lorsque nous sommes dans la ville *X*, il serait bon de connaitre *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).

Méthodes de recherche et de contrôle

Les valeurs de cette heuristique sont fournies dans la figure suivante. 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 heuristique était parfaite, nous n'aurions même pas besoin de faire une recherche !

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

Méthodes de recherche et de contrôle

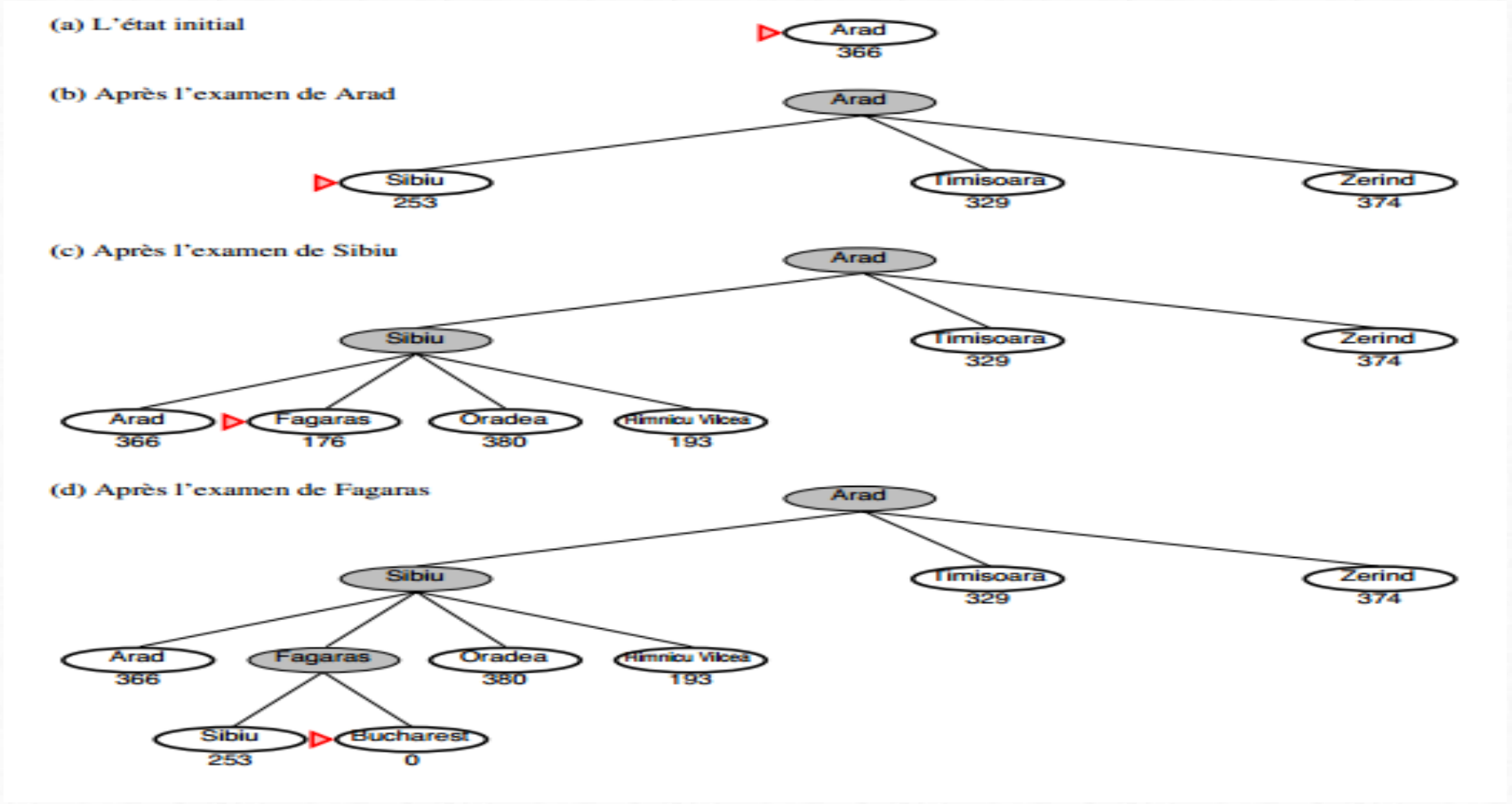
Best-first search (meilleur d'abord)

L'idée de l'algorithme "best-first search" (à ma connaissance, il n'y a pas de terme français couramment utilisé) est d'examiner les nœuds qui semblent les plus proches d'un état but, dans l'espoir d'aboutir plus vite à une solution.

Un peu plus formellement, la stratégie employée par best-first search consiste à utiliser la fonction heuristique h comme fonction d'évaluation (c'est-à-dire qu'on prend $f(n) = h(n)$).

Méthodes de recherche et de contrôle

La figure suivante montre le déroulement du best-first search sur le problème de trouver un chemin de Arad à Bucarest que nous venons de présenter.



.

Méthodes de recherche et de contrôle

Un peu plus formellement, la stratégie employée par best-first search consiste à utiliser la fonction heuristique h comme fonction d'évaluation (c'est-à-dire qu'on prend $f(n) = h(n)$).

La figure précédente montre le déroulement du best-first search sur le problème de trouver un chemin de Arad à Bucarest que nous venons de présenter.

Nous commençons dans l'état initial Arad, puis nous considérons les trois successeurs possibles: Sibiu (avec valeur 253), Timisoara (avec valeur 329), et Zerind (avec valeur 374).

Méthodes de recherche et de contrôle

Le nœud Sibiu a la plus petite valeur heuristique, c'est donc lui qui sera examiné ensuite. Comme Sibiu n'est pas un état but, nous ajoutons ses quatre successeurs à la liste de nœuds à traiter : Arad (366), Fagaras (176), Oradea (380), et Rimnicu Vilcea (193).

Le nœud le plus prometteur (parmi tous les nœuds restants à traiter) est Fagaras. Comme Fagaras n'est pas un état but, nous ajoutons ses deux successeurs Sibiu (253) et Bucarest (0) à la liste de nœuds à traiter. Nous choisissons alors Bucarest puisque ce nœud possède la plus petite valeur heuristique.

Comme ce nœud correspond à un état but, nous arrêtons la recherche et renvoyons le chemin trouvé (Arad, Sibiu, Fagaras, Bucarest).

Méthodes de recherche et de contrôle

Best-first search n'est ni complet ni optimal. Il n'est pas complet car l'algorithme peut tourner en boucle même s'il existe une solution.

Pour voir qu'il n'est pas optimal, il suffit de regarder l'exemple que nous venons d'examiner : le chemin (Arad, Sibiu, Fagaras, Bucarest) retourné par l'algorithme est plus long que le chemin (Arad, Rimnicu Vilcea, Pitesti, Bucarest).

Pour une profondeur maximale de p et s successeurs, la complexité en temps et en espace de best-first search sont toutes les deux en $O(s^p)$ dans les pire des cas, même si la complexité dépend en pratique de la qualité de la fonction heuristique.

Recherche A*

Best-first search donne la préférence aux nœuds dont les états semblent les plus proche d'un état but, mais il ne prend pas en compte les coûts des chemins reliant l'état initial à ces nœuds.

Néanmoins, c'est une information très pertinente car le coût d'un chemin passant par un nœud n est la somme du coût de chemin entre l'état initial et n et le coût du chemin reliant n à un état but. C'est cette idée qui est à la base de la 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.

Méthodes de recherche et de contrôle

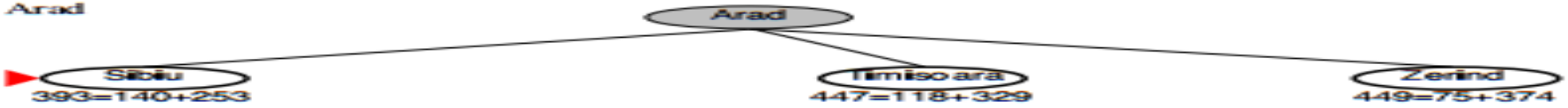
La figure suivante montre quelques étapes dans le déroulement de la recherche A* sur notre problème de trouver un chemin de Arad à Bucarest.

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 nœud à traiter : Arad (646= 280 + 366), Fagaras (415= 239 + 176), Oradea (671 = 291 + 380), et Rimnicu Vilcea (413= 220 + 193).

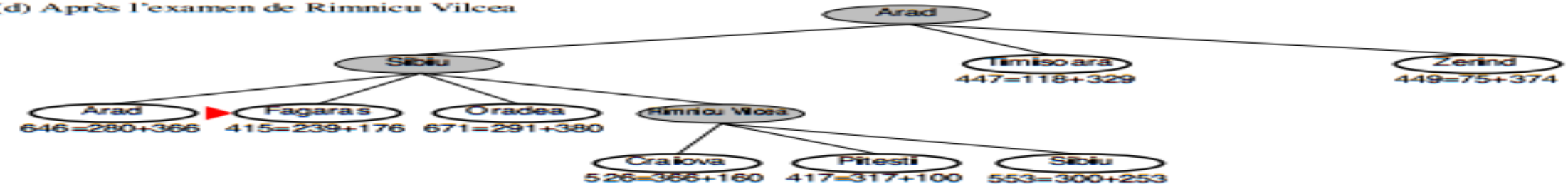
(a) L'état initial

(b) Après l'examen de Arad

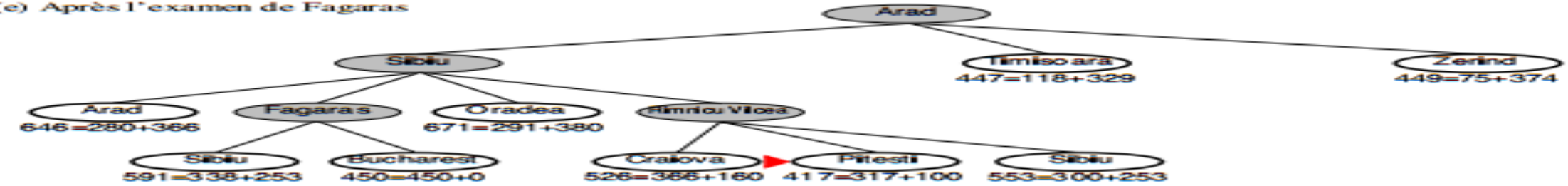


(c) Après l'examen de Sibiu

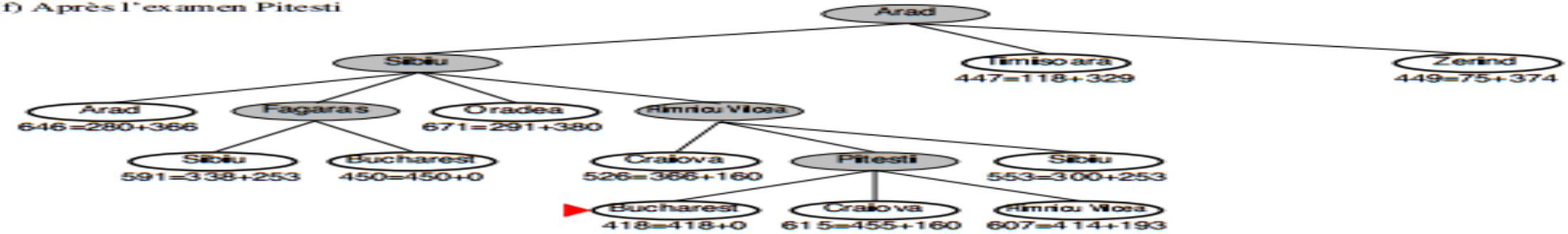
(d) Après l'examen de Rimnicu Vilcea



(e) Après l'examen de Fagaras



(f) Après l'examen Pitesti



AHI

Méthodes de recherche et de contrôle

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).

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.

Méthodes de recherche et de contrôle

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 nœuds 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.

Méthodes de recherche et de contrôle

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).

Méthodes de recherche et de contrôle

Si nous voulons utiliser la technique proposée pour éliminer les nœuds redondants (selon laquelle nous ne gardons que le premier chemin amenant à un état), il nous faut placer une restriction plus forte sur la fonction heuristique h pour garantir l'optimalité : la fonction doit être *consistante*.

Une fonction heuristique est consistante si pour tout nœud n et tout successeur n' obtenu à partir de 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.

Méthodes de recherche et de contrôle

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 comme iterated A* search (IDA*), recursive best-first search (RBFS), memory-bounded A* (MA*), et simplified memory bounded A* (SMA*).

Méthodes de recherche et de contrôle

Recherche Locale

Nous avons vu que pour certains problèmes, en particulier les problèmes de satisfaction de contraintes, nous ne nous intéressons pas au chemin reliant l'état initial à l'état but mais seulement à l'état but lui-même.

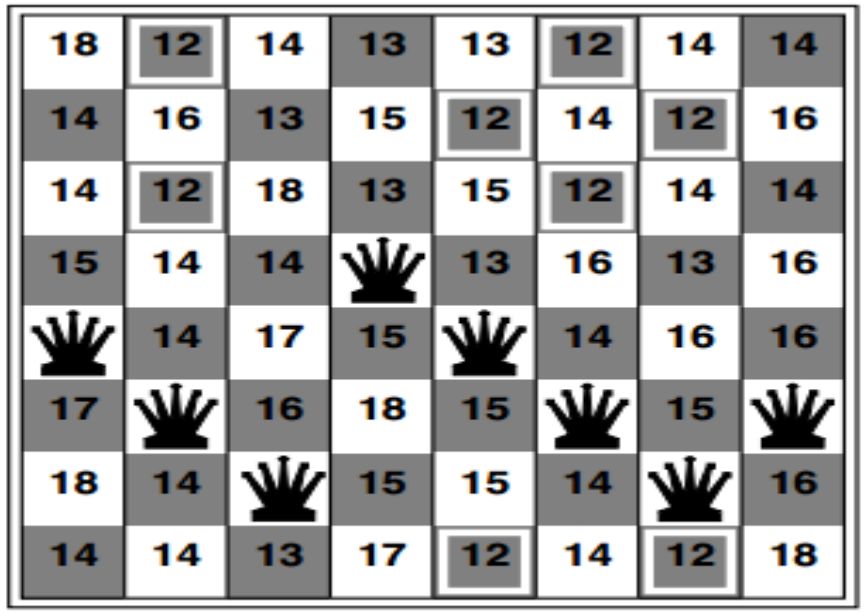
Pour les problèmes de ce type, il existe une autre stratégie possible : générer les états successivement sans s'intéresser aux chemins jusqu'à ce que nous trouvions un état but.

Cette idée est à la base algorithmes de recherche locale, qui sacrifient la complétude pour gagner du temps et de la mémoire. L'algorithme local le plus simple s'appelle la recherche locale gloutonne.

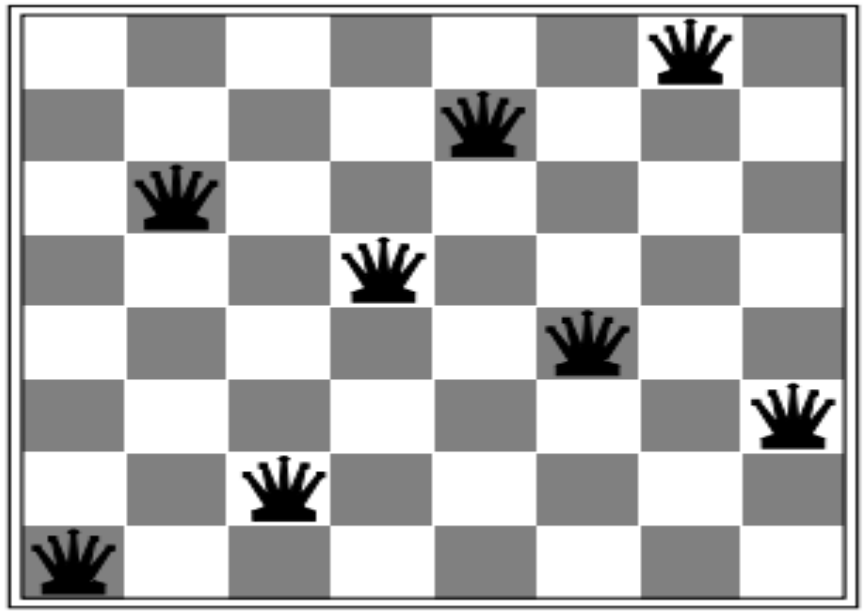
Méthodes de recherche et de contrôle

Nous commençons avec un état choisi aléatoirement. Si l'état est un état but, nous arrêtons la recherche. Sinon nous générons ses successeurs et leurs valeurs heuristiques(figure a). S'il n'existe pas de successeur avec une meilleure valeur que la valeur heuristique de l'état actuel, nous pouvons plus améliorer la situation, donc nous arrêtons la recherche (figure b). Sinon, nous choisissons l'état successeur ayant la meilleure valeur heuristique, et nous continuons ainsi.

.



(a)



(b)

Méthodes de recherche et de contrôle

- L'avantage de la recherche locale gloutonne est qu'elle a une très faible consommation en mémoire (il ne faut garder que l'état actuel en mémoire) et aussi en temps.
- Par exemple, pour le problème des huits reines, l'algorithme termine après 4 étapes quand il trouve une solution et 3 étapes quand il est bloqué (en moyenne).
- L'inconvénient principal de la recherche locale gloutonne est son faible taux de succès qui résulte du fait que nous arrêtons la recherche si jamais nous ne pouvons plus améliorer directement la valeur heuristique.

Méthodes de recherche et de contrôle

- Pour le problème des huit reines, cet algorithme ne réussit que 14% du temps. Il y a plusieurs améliorations possible de la recherche locale gloutonne.
- Une idée simple serait de permettre l'algorithme de visiter les successeurs ayant la même valeur que l'état actuel (il faut mettre une borne sur le nombre de déplacements consécutives de ce type pour ne pas tourner en boucle).
- Cette modification augmente significativement le taux de succès. Pour le problème des huits reines, si nous permettons au plus 100 déplacements consécutifs qui n'améliorent pas la valeur heuristique, le taux de succès passe de 14% à 94%.

Méthodes de recherche et de contrôle

- Une autre possibilité est d'ajouter de l'aléatoire : au lieu de choisir toujours le meilleur voisin, nous pouvons choisir un voisin aléatoirement (où la probabilité de sélectionner un état est défini en fonction de sa valeur heuristique).
- Finalement, nous pouvons tout simplement recommencer la recherche locale gloutonne à partir d'un autre état choisi au hasard, et continuer ainsi jusqu'à obtenir un état but.

Méthodes de recherche et de contrôle

- Cette technique pourrait sembler un peu simpliste, mais elle s'avère très efficace : elle permet de résoudre le problème des 3 millions de reines en moins d'une minute sur un ordinateur de bureau
- Grace à leur très faible complexité en espace, les algorithmes locaux peuvent être utilisés pour résoudre des problèmes de taille importante qui ne peuvent pas être résolus avec des algorithmes classiques.

Résolution des problèmes

Formalisation

Formalisation

Définition formelle d'un problème:

Un problème peut être défini par les cinq éléments suivants :

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

Formalisation

Nous pouvons voir un problème comme un graphe orienté où les nœuds sont des états accessibles depuis l' état initial et 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 citons des exemples concrets qui devraient nous permettre de mieux comprendre cette définition.

Formalisation

Exemples de problèmes

Nous commençons par deux problèmes ludiques :

- le taquin.
- le problème des huit reines.

Formalisation

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 les deux exemples dans le diapositive suivants.
- **Etat initial:** N'importe quel état pourrait être choisi comme l' état initial.

Formalisation

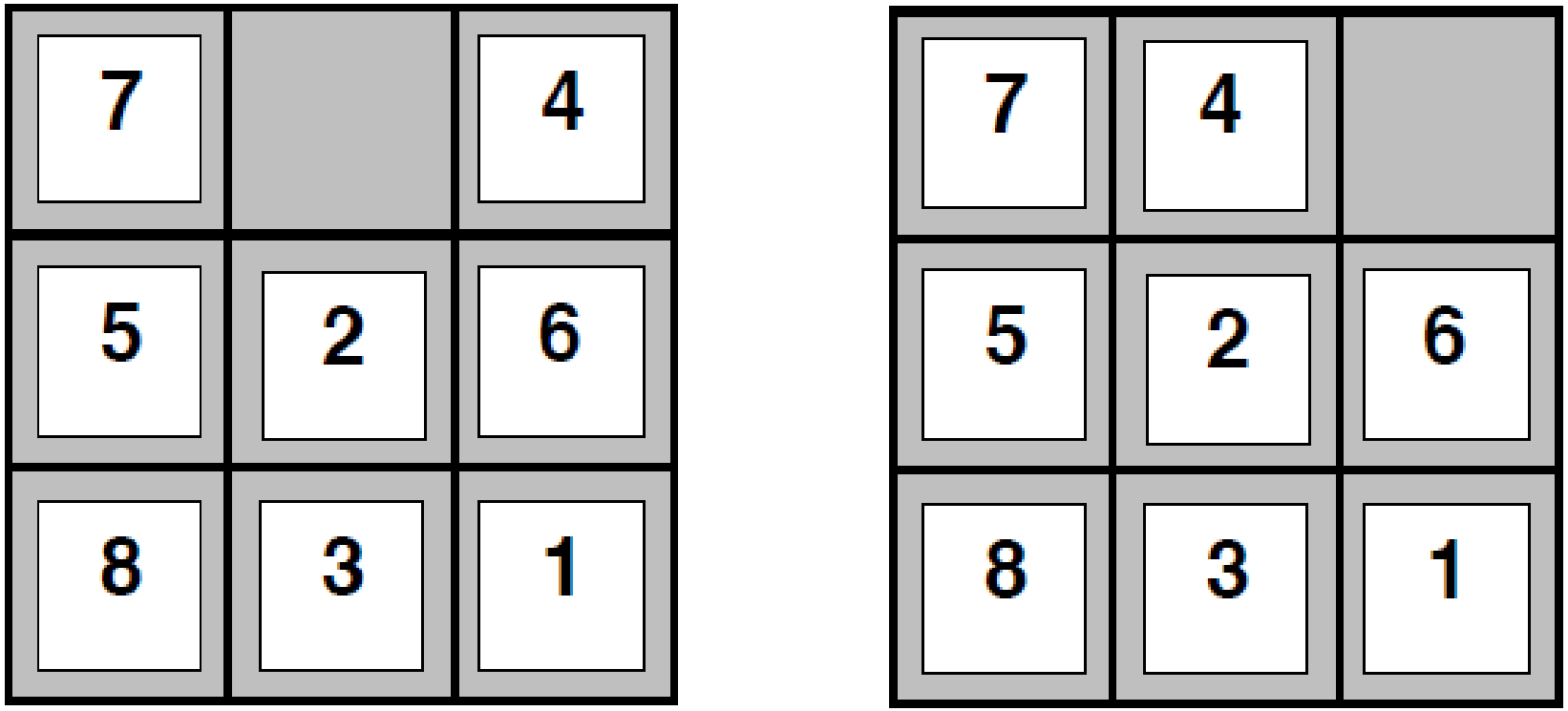


Fig. 1 – Deux états du jeu de taquin.

Formalisation

- **Actions:** Il y aura 4 actions possibles correspondant aux quatre façons de changer la position du carre 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 fixe 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).

Formalisation

- **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 3 x 3 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 (avec un espace d'états de taille 1025) restent difficiles.

Formalisation

Exemple 2 (Huit reines). L'objectif de ce jeu est de placer huit reines sur un échiquier (une grille 8 x 8) 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.

Formalisation

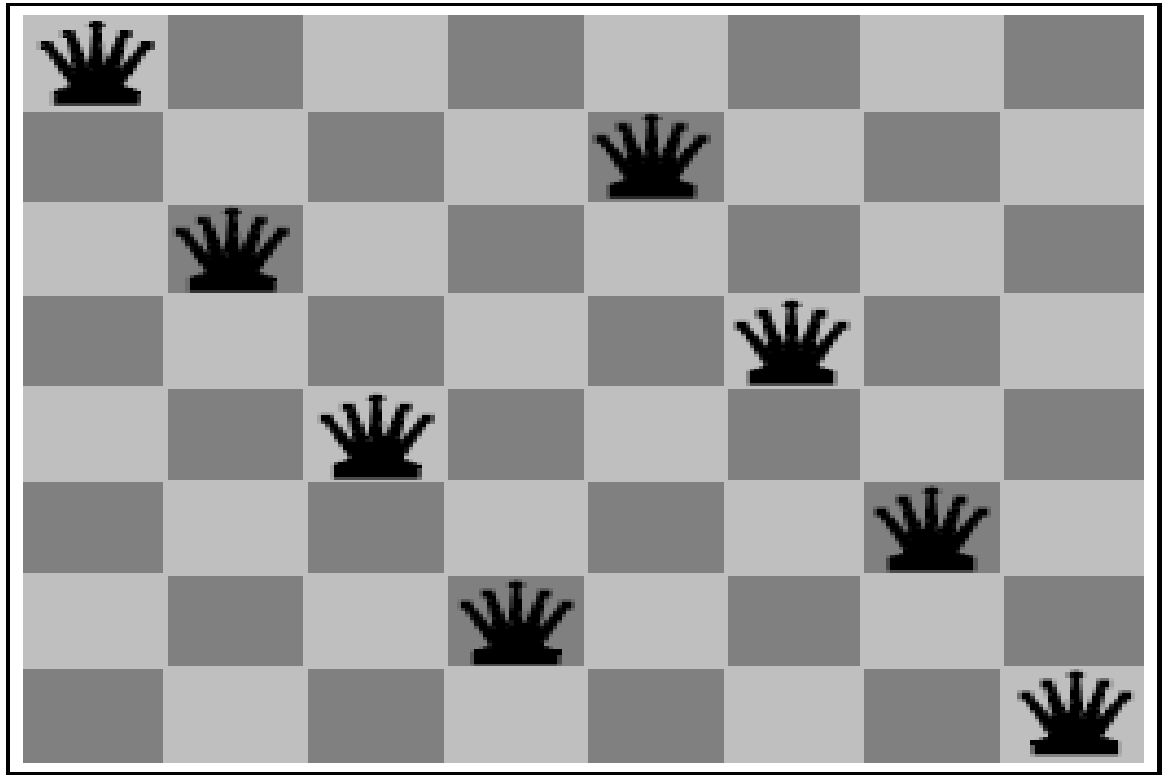


Fig. 2 – Une presque solution pour le problèmes des huit reines.

Formalisation

La 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 spécifie à la configuration courante.
- **Test de but:** Une configuration de huit reines avec aucune reine sous attaque.
- **Couts des actions:** Ce pourrait être 0, ou un coût constant pour chaque action - nous nous intéressons pas du chemin, seulement l' état but obtenu.

Formalisation

On peut réduire l'espace d'états drastiquement (de 3x10 puissance 14 à 2057 !) en observant qu'il est inutile de continuer de développer des configurations où il y a déjà un conflit (comme on ne peut pas résoudre le conflit en ajoutant des reines supplémentaires).

Les changements de notre formalisation sont:

- **Etat initials:** Configurations de n reines ($0 < n < 8$), une reine par colonne dans les n colonnes plus à gauche, avec aucun conflit.

Formalisation

- **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 huit reines sur la grille (une par colonne) et puis de changer la position d'une reine à chaque tour.

Formalisation

- **Etat initiaux:** 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 visite, 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 explore ou de tourner en boucle.

Formalisation

- Nous remarquons que ces deux problèmes sont de nature assez différentes. Avec le taquin, nous savons depuis le début quel état nous voulons, et la difficulté est de trouver une séquence d'actions pour l'atteindre. En ce qui concerne le problème des huit reines, nous ne sommes pas intéresses par le chemin mais seulement par l'état but obtenu.

- Ces deux jeux sont des exemples de deux grandes classes de problèmes étudies en IA : des problèmes de planification et des problèmes de satisfaction de contraintes. Nous examinons plus en détail ces deux classes de problèmes.

Formalisation

Exemple 3 (Problèmes de planification). Dans un problème de planification, nous cherchons une suite d'actions pour relier l'état initial à un état but.

Voici une formalisation standard du problème :

Etats: Un ensemble d'états du monde qui correspondent aux différentes valuations d'un ensemble de propositions. Par exemple, considérons un cas très simple avec une seule proposition allumée (qui dit que la lumière est allumée). Nous avons deux états du monde possibles : allumée=vrai et allumée=faux.

Formalisation

- **Etat initial:** L' état actuel du monde, e.g. allumée=vrai.
- **Actions:** Un ensemble d'actions autorisées, e.g. actionner l'interrupteur.
- **Fonction de successeur:** Cette fonction va nous dire quelles actions sont possibles dans chaque état du monde et comment les actions changent l'état du monde, e.g. l'action actionner interrupteur va faire passer de l'´etat allumée=vrai à l'état allumée=faux et inversement.

Formalisation

- **Test de but:** Théoriquement, ce pourrait être n'importe quel ensemble d'états, mais souvent nous définissons le but comme l'ensemble des états satisfaisants une certaine valuation partielle des propositions.
- **Coût des actions:** Pour essayer de trouver des plans les plus courts, nous pouvons mettre tous les coûts à 1. Nous pouvons aussi définir les coûts des actions en fonction de leur durée, de leur coût en argent, de la quantité de ressources qu'elles consomment, etc.

Formalisation

Les problèmes de planification sont courants dans la vie de tous les jours. Par exemple, considérons le problème de trouver le chemin le plus court pour atteindre une destination. C'est un problème tellement courant qu'il y a maintenant des GPS dédies à cette tâche. Citons également les systèmes de planification de voyage qui cherchent des itinéraires en transports en commun, en train ou en avion.

Formalisation

Formalisons du problème de planification de voyage :

Etas Chaque état est compose d'un aéroport et la date et l'heure actuelle.

Etat initial L'aéroport d'où part le client, la date de départ souhaitée, et l'heure à partir de laquelle le client peut partir.

Actions Ce sont des vols d'un aéroport à un autre.

Fonction de successeur Les actions possibles sont des vols qui partent de l'aéroport de l'état actuel, plus tard que l'heure actuelle (en fait, nous devrons aussi exiger une certaine durée entre l'arrivée dans un aéroport et le prochain vol, sinon le client risque de rater sa correspondance ! !).

Formalisation

Test de but Les états où le client est à l' aéroport de sa destination.

Cout des actions Cela va dépendre des préférences du client. Ce pourrait être 1 pour chaque action (pour minimiser la nombre de connections), ou la durée des vols (pour minimiser la durée du voyage), ou le prix des trajets (pour trouver le voyage le moins cher).

Un problème très similaire est celui du routage dans les réseaux, ou il faut trouver les chemins pour envoyer des paquets.

Enfin, les planificateurs sont aussi utilises dans l'industrie pour trouver des manipulations physiques permettant à un robot de construire des objets complexes à partir de composants donnes.

Formalisation

Exemple 4 (Satisfaction de contraintes)**.** Un problème de satisfaction de contraintes (CSP) consiste en un ensemble de variables, des ensembles de valeurs permises pour chaque variable, et un ensemble de contraintes (sur les combinaisons des valeurs des variables). L'objectif est de trouver une valuation telle que chaque contrainte est satisfaite. Voici une formalisation incrémentale d'un CSP :

Etat initiaux: Des valuations partielles (c'est à dire, des choix de valeurs pour un sous-ensemble des variables)

Etat initial La valuation vide (où nous n'avons pas encore choisi des valeurs)

Formalisation

Actions Choisissez une valeur pour une des variables restantes.

Fonction de successeur Nous ajoutons la nouvelle valeur à la valuation actuelle.

Test de but Un état but est une valuation complète telle que chaque contrainte est satisfaite.

Coût des actions Cela peut dépendre du problème en question, mais en général nous devrons donner le même coût pour chaque action comme les solutions sont toutes aussi bonnes.

Formalisation

Nous pouvons également donner une formalisation à partir des états complets :

Etats Une valuation des variables.

Etat initial Une valuation quelconque.

Actions Changez la valeur d'une variable.

Formalisation

Gérer des pistes d' atterrissage à l' aéroport, trouver un plan de table pour un mariage, créer un emploi du temps à l' université ce sont tous des problèmes de satisfaction de contraintes.

Pour ce dernier exemple, les variables sont des cours, des valeurs possibles sont des triplets (jour, heure, salle), et des contraintes de toute sorte (il faut qu'un cours INF100 soit enseigne le vendredi, il faut pas mettre INF101 et INF102 en même temps, etc.).

Formalisation

Nous donnons une formalisation incrémentale de cet exemple :

Etats Un emploi de temps partiel (c'est à dire, un choix d'une triplet (jour, heure, salle) pour un sous-ensemble des cours).
Etat initial Un emploi de temps vide.
Actions Choisissez une triplet (jour, heure, salle) pour un cours qui n'est pas encore sur la grille.
Fonction de successeur L'emploi de temps résultant.
Test de but Un emploi de temps contenant tous les cours et satisfaisant toutes les contraintes.
Coût des actions Un coût constant.

Formalisation

Dans tous ces problèmes considères jusqu'à présent, il n'y avait qu'un seul agent qui pouvait choisir librement ses actions. Mais souvent nos décisions sont conditionnées par les actions des autres. C'est notamment le cas pour les jeux à plusieurs joueurs, que nous considérons maintenant.

Exemple 5 (Jeux)**.** Les jeux à plusieurs joueurs sont plus compliques que les problèmes que nous avons vu parce que l'agent ne peut pas choisir les actions des autres joueurs. Donc, au lieu de chercher un chemin qui relie l'état initial à un état but (qui contiendrait des actions des adversaires, dont nous n'avons pas le contrôle), nous allons chercher une *stratégie*, c'est à dire un choix d'action pour chaque état où pourrait se trouver l'agent.

Formalisation

Voici une formalisation d'un jeu à plusieurs joueurs :

Etats Des configurations du jeu plus le nom de joueur dont c'est le tour.

Etat initial La configuration initiale plus le nom du joueur qui commence.

Actions Les coups légaux pour le joueur dont c'est le tour.

Fonction de successeur La nouvelle configuration du jeu, et le nom de joueur dont c'est le tour.

Test de but Les configurations gagnantes pour le joueur.

Coût des actions Cela dépendra du jeu en question.

Formalisation

Pour le jeu d'échecs, nous aurions la formalisation suivante :
Etats Des états sont composés d'une configuration de l'échiquier plus le nom du joueur dont c'est le tour.
Etat initial La configuration standard pour le début d'une partie d'échecs.
Actions Ce sont des coups qui peuvent être joués par le joueur dont c'est le tour dans la configuration actuelle du jeu (y compris l'action d'abandonner le jeu).
Fonction de successeur La configuration du jeu évolue selon le coup choisi, et c'est maintenant à l'autre joueur de jouer.

Formalisation

Test de but Il y a beaucoup de façons de terminer une partie, la plus connue étant l'échec et mat.

Coût des actions Nous pouvons les mettre à 1 pour chercher les coups qui amènent le plus vite à une configuration gagnante.

Formalisation

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 **echec**

– 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.

Ce qui va différencier les différents algorithmes est la manière dont on effectue le choix de l'étape.

Évaluation des algorithmes de recherche

Dans la suite, nous allons voir différents d'algorithmes de recherche. Comment pouvons-nous les comparer ?

Les quatre critères que nous allons utiliser pour comparer les différents algorithmes de recherche sont :
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 ?