IA- Matthieu Exbrayat

Alexandre Masson

14 Janvier 2013

Table des matières

1 Organisation du cours

Cours de 2h le mardi et TD 2 h le mercredi.

2 Introduction

Questions Qu'est ce que les taches suivantes ont en commun ?

Concevoir des systèmes capables de faire des choses compliquées.
Il y a aussi l'aspect ; apprentissage , le système est il capable d'apprendre de lui même et se débrouille tout seul.
Exemples :
Concevoir un programme qui vire tout seul les spam dans les mails.
Concevoir un super navigateur qui s'occupe tout seul de faire les mise à jour logicielle.

Le point commun ? Posséder un certain degré d'intelligence.
D'où grande question qu'est ce que l'intelligence ?

- Selon Darwin : Ce qui permet l'individu le plus apte, parfaitement adapté a son environnement
- Selon Edison : ce qui fonctionne et qui produit de l'argent.
- Selon Turing : ce qui rend difficile la distinction entre une tache réalisée par un être humain ou une machine

L'IA est une discipline qui systématise et automatise les taches .

- Penser comme un humain
- Agir comme un humain
- Penser rationnellement
- Agir rationnellement

Penser comme un humain Il faut comprendre l'esprit Humain.
Comparaison des différentes étapes d'un programme et du raisonnement humain pour arriver au même problème.
Science cognitive.

Agir comme un humain Test de Turing, on transforme l'ordinateur peut il penser en peut il se comporter intelligemment. Le test est concluent si l'opérateur ne sais pas si la réponse a ses question est donnée par un humain ou une machine

Penser rationnellement : approche logique

Agir rationnellement : agir pour atteindre un objectif Remplir une mission de la meilleure façon.

Agent : unité qui fonctionne de façon autonome, perçoit son environnement , s'adapte aux changements et est capable d'atteindre un objectif.

Agent Rationnel : agent qui agit pour atteindre le meilleur résultats ou du moins le meilleur résultat espéré.

Loebner prize Tous les ans récompense le meilleur système du test de Turing.

- traitement du langage naturel : pouvoir communiquer en un langage naturel.
- représentation de connaissances : stocker ce qu'il sait ou ce qu'il perçoit.
- raisonnement automatique : utiliser des connaissances pour répondre aux questions.

les Lois de la pensée Aristote : quels sont les processus de pensée corrects ?
Plusieurs formes de logique : notation et règles de dérivation pour les pesées , sans relations avec une mécanisation du raisonnement.
Ligne directe entre math et philo -> IA.
Problèmes : tous les comportements intelligents ne sont pas le résultats d'un raisonnement logique.

Agent Rationnel abstrait : un agent est une fonction qui met en correspondance des séquences perceptives P* et des actions A : f : P* -> A.

processus où on ne sais pas tout, ou temps limité trop court pour répondre, donc réponse acceptable mais pas optimale.

Fondements de l'IA Elle repose sur plusieurs domaines.

Philosophie : logique et raisonnement.

Mathématiques : représentations formelles et preuves, algorithmes, (in)décidabilité, probabilité.

Psychologie : adaptation, perception et controle moteur, techniques expérimentales.

Linguistique : représentation de connaissances grammaires.

Neurosciences : substrat physique et biologique de l'activité mentale.

Théorie du controle : systèmes asservis, stabilité, concept d'agent optimal.

3 Histoire de l'IA

- 1943 : modélisation de neurones.
- 1950's : vision complète de l'IA
- 60's : dev des réseaux de neurones.
- 70's : développement des systèmes à base de connaissances, faire de l'aide au diagnostic.
- fin 80's : hiver de l'IA, l'ia s'effondre
- depuis 20 ans : IA évolue et révolutionne sa méthodologie.
- plein de fric à se faire dans la Bourse, algorithme qui réagissent au plus vite et essaye d'analyser la bourse pour établir des règles.

Prédictions et réalité différence prédictions réalité ;, 60's œil électronique, pas encore fait mais ça avance et s'est encourageant.

Robot qui font tout, déjà dans les 70's.

4 Qu'est ce qu'un problème pour l'IA

Problème qui n'as pas de solution analytique connue, objectif : si l'objectif est hors du possible donner une solution acceptable en temps raisonnable.

Certaines taches aisées pour l'humain sont difficiles pour la machine : tout ce qui a besoin de la sensibilité humaine.

Contenu du cours 4 approches différentes de la résolution de problèmes en IA

- recherche de solution dans un espace d'états
- recherche par raisonnement et en présence d'incertitude
- résolution par planification.
- résolution par apprentissage automatique

5 Agents Intelligents

Qu'est ce qu'un agent L'AGENT perçoit des choses de son ENVIRONNEMENT, à laide de SENSORS, il accompli ensuite sa MISSION, et utilise ses ACCUATORS, pour effecteur des actions.

Définition : entité capable de percevoir son environnement par des capteurs et d'agir sur son environnement à l'aide d'effecteurs (actionneurs). AUTONOMIE , PERCEPTION,ACTION,OBJECTIF, sont les maitres-mots qui définissent l'existence de l'agent.

spécification d'un agent le choix d'une action à l'instant t dépend de séquences perceptives .

Agent rationnel Basé sur le raisonnement.

Toute option considérées , faire le meilleur choix pour maximiser les chances de succès.

Mesures de performances : réussir la tâches, quantification maximale d'un objectif.

spécifier l'environnement spécif = problème, agent) solution

PEAS exemple , conduite automatique
- P (mesure de performance) : bon endroit, temps, sécurité
- E (environnement) : rues, voitures, piétons, météo
- A (actions) : tourner , arrêter , klaxonner, etc...
- S (senseurs) : plein.

types d'environnement Observable ou nono , et déterministe ou non., Observable : on a accès à tout moment à l'état complet de l'environnement. Déterministe : on connais l'instant suivant à partir de l'état courant et l'action faite par l'agent.

Épisodique/séquentiel : les états futurs ne dépendent pas des actons effectuées lors de évents précédents.

Statique/dynamique : un environnement ne change pas durant le temps ou l'agent réfléchit.

Discret/continu : environnement discret si le nombre de séquences est fini. Simple agent ou multi-agents. les appli du monde réel sont les plus difficile à mettre en place car l'environnement n'est que partiellement observable, il est séquentiel, dynamique, continu et multi-agent.

types d'agents

- Agent-reflex : presque tout est codé en dur, on applique des règles en fonction de se qu'on observe de l'environnement.
- Agent-reflex avec états : on va conserver des états qui vont servir à estimer des états en fonction de se qu'on viens de capter et ainsi que ce que l'on a sauvegardé. Mettre a jour l'état interne dépend de comment l'env change et comment nos action le font évoluer.
- Agent avec objectif. le Système va être dans un état donné, et après on regarde toutes les solution dispo et on va choisir celle qui nous rapproche le plus de l'objectif. IL a un objectif que décrit les situations désirées. il combien ces informations avec la résultat pour choisir la bonne action. l'agent devrait être capable de considérer des longues séquences pour atteindre l'objectif : **recherche** et **planification**.
- Agent avec la fonction d'utilité : parmi la listes des actions proposées celle qui est la plus prometteuse. On est déjà sur des solutions sophistiquées. Il n'est pas capable de s'améliorer lui même.
- Agent d'apprentissage : il reçoit des signaux et choisis quoi faire et indique ces décisions. il va ensuite essayer de savoir si ça décision est pertinente. Il a des exemples d'actions et de suites d'actions qu'il a effectuées, il sais si ces actions sont bien choisies ou pas

IL est nécessaire d'etre capable de stocker des processus qui vont prendre des décisions. La vision PEAS est une modélisation simple.

6 Problèmes et algorithmes de recherche

6.1 Agents de résolutions de problèmes

Intro aux algorithmes de recherche Les algorithmes de recherche constituent l'une des approches les plus puissantes pour la résolution de problème en IA.

Les algo de recherche sont un mécanisme de recherche général de résolution. On sais toujours tout ce qui peux être fait depuis cet état mais on ne peux pas savoir à l'avance si cet état en bon ou pas. On effectue à chaque action un test de solution. On sais a quoi ressemble un état qui peux être une solution, et il est aussi intéressant de savoir comment nous sommes arrivé a la solution. Plusieurs approche possible en largeur ou en profondeur. Exemple : voyage d'un bout à l'autre de la Bulgarie.

Autre exemple : jeu de taquin : puzzle-8 : on a des cases dans un ordre aléatoire, mais on veux arriver a avoir toutes les cases dans l'ordre.

Définition d'un problème de recherche Formalisons un ptit peu ça : tous les états sont dans un ensemble Q : non vide, il y a parmi ceux la des états initiaux(S). On a aussi un ensemble d'état solutions(G) : définis explicitement, ou implicitement avec des règles de tests. A : un ensemble d'actions . Fonction de successeurs, et parfois une fonction de coût : cela nous permet de pouvoir estimer si une décision est bonne ou pas. Nous avons besoin de définir tous ces éléments pour définir les problèmes de recherche. Exemple :
Aspirateur : problème assez simple. Solution : séquence d'opération .

Il existe aussi des problèmes plus compliqué , les problèmes contingents : effet conditionnel des actions, perception fournit des nouvelles infos de l'état courant, la solution est un arbre ou une stratégie. IL existe aussi les problèmes d'exploration : l'exécution révèle les états, besoin d'expérimenter pour trouver la solution.

6.2 Formulation d'un problème à état unique

Un problème est défini par 4 éléments :
état initial : être à Arad.
Fonction de successeur.
test : implicite : NoDirt(x), ou explicite : "être à Zerind". Tous les problèmes qui peuvent être décrits par : un ensemble fini d'états, un ensemble fini d'actions, un sous-ensemble d'états initiaux et solutions, une relation successeur définie sur l'ensemble des actions et qui renvoie vers l'ensemble des états et une fonction de coût qui est positive.

6.3 Stratégies

Elles sont évaluées sur 4 critères : Complétude : si une solution existe, le système va t il l'atteindre.
complexité en temps : combien de nœuds on est censé explorer. Deux grandes familles, aveugles et heuristiques.

Stratégies Aveugles On utilise la recherche en profondeur limitée, pour profiter de la vitesse de calcul de la largeur, mais aussi de l'utilisation mémoire plus satisfaisante de la recherche en profondeur. Cette solution est complète, sa complexité en temps est $O(b^d)$, sa complexité en espace est O(bd). Optimalité, si le coût pour passer d'une étape à une autre est unitaire, car toues les solution d'un même niveau ont le même coût.

Arbre de recherche bidirectionnelle nécessite de connaître explicitement les états finaux, et des actions sont on connais la fonction inverse, on pars de la source et des solutions, et on construit en fonction de la longueur des chemins tous les nœuds accessibles, et on regarde s'il y a des matchs.

Quelle solution prendre ? Si on a pas trop de problème de mémoire , le bidirectionnel est le plus efficace, si on a des contraintes de mémoire, on aura plus intérêt a prendre la recherche par profondeur itérative.

États répétés L'échec de détection d'états répétés risque de produire des arbres de recherche infinis.
En largeur, garder une trace de tous les états déjà parcouru, si l'état d'un nouveau nœud existe déjà , alors on le supprime.

Recherche heuristiques Nous allons utiliser une fonction heuristique , de l'ensemble des états vers les réels comme une estimation du rapport coût bénéfice qu'il y a a étendre le chemin courant en passant par s.

Idée : Utilisation d'une fonction d'évaluation (heuristique) pour chaque nœud, étendre avec le plus prometteur selon la fonction heuristique. On utilise souvent l'approche gloutonne pour minimiser le coût des transitions. Cette approche gloutonne est elle complète ? : il existe des cas où l'on boucle., mais complet dans un espace fini sans boucle.

En terme de temps , c'est de l'ordre , facteur de branchement avec en exposant la profondeur maximum, pareil en espace, car il faut garder en mémoire tous les nœuds. La performance de cette approche gloutonne, dépend de la précision de la fonction heuristique, il est possible de réduire fortement les contraintes de temps et d'espace.

Une fonction heuristique est admissible (qui ne surestime jamais le coût réel si elle est supérieure a 0 , et inférieur au coût optimal réel, une fonction heuristique est toujours optimiste !

L'idée va être de combiner le greedy search(glouton), réduisant le coût , mais pas toujours optimal, avec le coût uniforme, qui lui est optimal, mais pas très efficace, pour arriver a une solution admissible.

Cela nous mène a l'algorithme A*, éviter de choisir le chemin qui est déjà coûteux, on ne va plus uniquement regarder vers l'avant , mais aussi vers l'arrière, pour prendre n compte le chemin qui mène a un nœud, en plus de sa capacité a nous rapprocher du but.

Complétude du Lemme Oui , sauf dans le cas ou on a un facteur de branchement infini,
Temps Exponentiel,
Espace Exponentiel, il faut garder tous les nœuds en mémoire,
Optimalité Oui , f_{i+1} n'est pas étendu tant que f_i n'est pas fini.
A* étend tous les nœuds ayant f(n) < C*
A* étend quelques nœuds ayant f(n) = C*
A* n'étend aucun nœud ayant f(n) > C*

Qualité de fonction heuristique Facteur de branchement effectif, soit N le nombre total d'états produits pour obtenir la solution, soit d la profondeur à laquelle la solution a été trouvée alors b* est la facteur de branchement d'un arbre fictif parfaitement équilibré tel que

$$N = 1 + b^* + (b^*)^2 + ... + (b^*)^d$$

Une bonne fonction heuristique aura une valeur de b^* proche de 1 (la fonction heuristique idéale aurait $b^* = 1$)

Dominance Si $h_2(n) >= h_1(n), \forall n$ (les deux étant admissibles) alors h_2 domine h_1 et constitue une meilleure fonction heuristique.
problème simplifié Une heuristique admissible peut être dérivée à partir du coût exact de la solution d'une version "simplifiée" du problème
Point clé Vérifier que le coût optimal du problème simplifié n'est pas plus grand que le coût du problème initial.

Variantes de A* Les problèmes réels sot souvetn très complexes
L'espace de recherche devient très grand
Meme les méthodes de recherche heuristique deviennent inefficaces
A* : complexité en espace exponentiel.
IDA* = approfondissement itératif de A*
SMA* = A* avec gestion mémoire.

IDA* IDS effectue l'approfondissement par rapport à la profondeur de la recherche
IDA* utilise comme limite la la valeur de la fonction heuristique f(n), chaque itération de IDA* est une recherche en profondeur, la profondeur limite correspond à la plus petite valeur observé au delà de la limite de l'étape précédente.
IL est complet et optimal
Complexité en espace de IDA* : O(bd), ok pour coût unitaire, mais qu'en est il pour des coûts réels ?

Un alternative : RBFS
Récursive BFS
Approche best-first, mais
Enregistre dans chaque nœud le f du meilleur chemin alternatif

remonte si le cout est supérieur à un f mémorisé
En remontant, mémorise ce cout (pour redescendre si besoin)
+ efficace qu'IDA* mais risque de redévelopper beaucoup de nœuds
Optimale (si h est admissible), espace O(bd) , temps ?
PB : la faible complexité en espace est finalement gênante.

SMA* Simplified Memory bounded A*
SMA* effectue sa propre gestion de mémoire
principe
- si la mémoire (file d'attente) est pleine, alors faire de la place en éliminant le nœud le moins intéressant (celui avec une valeur f élevée)
- retenir dans le nœud ancêtre la valeur du meilleur descendant oublié

Propriétés
- espace mémoire alloué par avance
- évite les états multipliés
- complet si espace mémoire suffisant

7 5 Mars 2013

Les algos de recherche locale sont adaptés aux pbs ou on cherche une solution mais on ne s'occupe pas du chemin. Ils considèrent un état courant, puisse déplacent successivement vers un état voisin par relation de voisinage.

propriété il sont : non-complets, non-optimaux, et non-exhaustifs, MAIS, utilisent peu de mémoire, souvent en quantité constante, ils conviennent aux recherches "offline" et "online".
Ils trouvent souvent des solutions raisonnables dans des espaces d'états très grands, voir infini, où les autres algos ne peuvent pas s'appliquer.
ils s'adaptent aux problèmes d'optimisation où l'on cherche le meilleur état suivant une fonction d'objectif.
un voisinage s'obtient en perturbant légèrement un état, on définit une distance $V(s) = \{s' | d(s, s') < \epsilon\}$.
on considère une fonction de cout qui mesure l'écrat entre l'état et l'objectif ou simplement la valeur de l'état courant, on cherche à optimiser cette fonction.

Exemple : TSP
L'état initial est un tour quelconque , puis les états voisins sont obtenus successivement en échangeant deux déplacements. des variantes de cette méthode trouvent une solution raisonnable rapidement avec des centaines de villes.

hill climbing (approche par escalade)
le principe consiste à rechercher , parmis les voisins , celui qui améliore le plus al fonction de cout, on recommence jusqu'au max, souvent max local, l'algo est glouton.

Satisfaction de clauses booléenne problème NP-Complet, distance de Hamming, nombre de bits différents. cout : nombre de clauses satisfaite par ce changement de bit. on pondère avec le nombre de variable, car les grandes clauses sont plus difficiles a satisfaire. l'algorithme peux osciller dans un maximum local, s'il il n'y a pas de réponse, on peut relancer l'algorithme. pour le climbing, il faut pouvoir sortir des plateaux (même valeur plusieurs fois), mais avec mouvement random, risque de boucler sur un plateau.

le recuit simulé idée, autoriser de redescendre vers un état moins bon avec une certaine probabilité.
La probabilité de descente dépend d'un paramètre appelé *température*
Au cours de la recherche, on diminue progressivement la température.
température élevée : recherche + aléatoire -> exploration de larges zones de l'espace de recherche
recuit simulé = simulated annealing.

local beam search garder K successeurs au oieu d'un , pas le meme comportement avec k recherches en parallèles, souvent k états finissent par se retrouver dans un maximum local. Idée : choisir k successeurs aléatoirement , *stochastic beam search* Analogie dans la sélection naturelle.

8 Les algorithmes génétiques

le principe maintenir une population de solutions candidates appelées individus(= états).
coder chaque individus (ses gènes).
générer au hasard une population initial.
exécuter le cycle suivant jusqu'à obtention d'un critère d'arrêt : Sélection -> reproduction -> mutation

9 Jeux

intro jeu constitue première branche de l'ia, premier sujet, les échecs, dans certains jeu, les algo actuel surpassent les humains. Exemple : deep blue : projet de 1985, deux séries, 300M de calcul par secondes, profondeur de 12 demi-coups de profondeur. Théorie des jeux, plusieurs participants (ou adversaires ou agents), principe de coop ou rivalité, en IA :

- a la basse , jeux alternés, déterministe, a somme nulle (ce que l'un gagne, l'autre le perd).
- notion de gain
- contrainte de temps.

de quoi dispose t on état initial, fonction successeur(coup,état) -> notion de coup légal, test de terminaison(-> états terminaux), fonction d'utilité(ou

de gain)(associe un score a chaque état, généralement associé au décompte des scores spécifique).

Les joueurs jouent à tour de rôle, un cou p = 2 demi-coup, le score et donné par joueur, l'un des joueurs essaie d'avoir le plus de score, et l'autre le moins.

mini-max stratégie optimale, reposant sur un exploration complexe : idée : trouver à chaque étape, le coup assurant un gain maximum (c'est a dire limitant les possibilités de gain de l'adversaire. On donne un score à chaque noeud et à chaque voisins pour choisir le meilleur. mini-max et multijoueur, on doit tenir compte de de l'intérêt de tous les joueurs, les résultats des fonction utilité, sont des vecteurs, ou chaque élément représente un des joueurs, alors lors des comparaisons pour le choix d'une action, chaque joueurs regarde le composante du vecteur qui le concerne et choisit le plus haut. **Bilan** Exploration complète, parcours en profondeur, coût en temps $O(b^m)$, coût en espace en $O(bm)$, adapté au problème simples. Peut on éviter certaines branches ?

Alpha-bêta Certain sous arbres sont inintéressant pour le joueur courant :e.g : on sais qu'il existe une super opportunité pour l'adversaire. technique élagage, suppose d'avoir des certitudes sur l'intérêt d'une branche -> génération d'au moins une feuille, évaluation de l'intérêt ?. en gros, si on trouve un sous arbre moins intéressant, alors on l'explore pas (soit max moins grand , soit min plus grand). **bilan** efficacité dépend de l'ordre des successeurs. Amélioration : table de transposition, souvent plusieurs chemin mènent a un même état , le premier passage permet de faire l'élagage. Mémorisation des états (les plus intéressants dans une table de hachage(pas besoin de recalculer les successeur d'une config déjà connue, car on connais déjà sa valeur d'utilité.

décision imparfaite en temps réel Dans la plupart des jeux il n'est pas possible de déplier tous les coups possibles. il faut jouer le mieux possible ne temps raisonnable, fonction d'évaluation, test d'arrêt. Fournit une estimation de l'utilité espérée pour une position donnée, fonction heuristiques. Critère : pas trop pourrie, temps de calcul maîtrisé (meme si on explore pas tout, l'intérêt doit être calculé vite, elle doit rendre le même résultats que la fonction d'utilité pour les états terminaux. Principe : description de l'état : attributs, première approche : classification, états -> classes = indexation. info statistiques par classe(e.g % de victoire, de défaite, de nul), valeur espérée : somme pondérée (pondération = score associé). risque d'avoir de nombreuses catégories.
une approche plus réaliste : fonction linéaire des attributs : on accorde un poids a chacun des attributs, et on fait la somme pondérée. Dans cette formule, on suppose l'indépendance des attributs.
Test d'arrêt : reprise de l'approche de alpha-bêta ne remplaçant le test d'état terminal par un test cut-off, suppose de gérer la profondeur courante de et fixer la profondeur d de cut off., autre approche : exploration itérative(ids) , meilleur solution atteinte lorsque le temps est écoulé. **Améliorations** Recherche d'états

stables cut off , éloignées de possibilités d'inversement de valeurs.

Hasard les possibilités de coup pour soi ou pour l'autre est non déterministes, on ajoute des noeuds intermédiaire pour .

10 19 Mars 2013

10.1 Agents à base de connaissances

L'agent doit être capable de déduire le bonnes actions, on a déjà vu et déjà travaille avec des algos pour trouver de bonnes solutions en utilisant des méthodes de recherche on a des description de initial a final et l'objectif est de trouver un chemin entre les deux, mais on est toujours dans un espace de recherche avec des actions pour arriver. on evolue dans un monde reel, il peut etre statique ou en evo permanente, Comment representer des connaissances ou des lois sur le monde reel, comment raisonner sur ces lois, et quelles représentations pour trier ces informations.

Exemple :chasse au Wumpus les cases peuvent etre vide, avoir un puit, ou un monstres, de l'or, l'objectif du jeu est de trouver l'or. le but est de trouver l'or dnas le moins de mouvements possibles, (on ne considère pas les diagonales), on sens un courant d'air dans les cases voisines des puits et une mauvaises odeur autour du monstre. on ne connais pas l'environnement

Description PEAS du monde du Wumpus Performance

- trouver l'or +1000
- mort -1000
- -1 par deplacement
- -10 décocher une fleche

Environnement (inconnu à l'avance)

- odeur désagréable dnas les cases adjacente au wumpus
- breeze a coté des puits éclat dans la case contenant de l'or
- fleche tue le Wumpus si on lui fais face
- grille limitées
- environnement mono-agent

Action

- tourner droite, gauche ou avancer
- saisir le trésor et déposer le trésor
- décocher flèche

Sensor

- breeze, odeur,éclat

l'agent doit être capable de représenter des états, actions, etc..
incorporer de nouvelles séquences perceptives
mettre a jour ces représentation du monde
déduire les propriétés cachées du monde
Déduire les actions appropriées

Ici, notre environnement est statique

suite de l'exemple dans la case de départ, l'agent ne sens rien , il sais donc lque les voisins sont safe, il va donc avancer, si il sent quelque-chose, alors il va arrêter son exploration et explorer un autre chemin, si il sens ailleurs l'odeur , alors proximité du monstre. meme si le case adjacentes au monstre laissent une odeur et celles des puits laisse du breeze, on ne peut sentir ce vent que si on est déjà sur la case adjacente
Dans chaque cas, l'agent déduit une conclusion à partir des informations disponibles, la conclusion doit être correcte si les informations sont correctes

systèmes expert utilise des connaissances spéc pour donner des conseils ou solutions
les connaissances d'un domaines sont données dans un base de connaissances
simuler le raisonnement de l'expert humain , moteur d'inférences
la perf du système ne dépend pas du moteur, mais plutôt des connaissances du systèmes
un système doit être riche : Knowledge is Power
comme un esprit humain, un système expert se spé dans un domaine, enrichit ses connaissances avec des expériences.

domaines des systemes experts Interprétation : former des conclsions de haut niveau à partir de données brutes
prédiction : trouver des conséquences probables des situations donénes
diagnostique : déterminer la cause du mal fonctionnement dnas des situation complexes à partir des symptomes observables
Configuration : construire une configuration des composants pour atteindre des objectifs de performance tout en satisfaisant des contraintes de configuration.
planification : déterminer une suite d'action pour arriver à un ensemble d'objectifs.
surveillance : comparer des comportements d'un système par rapport au comportment souhaité
controle : controler le comportement d'un environnement complexe

architecture bases de connaissances : générales et spécifiques, sous forme de règles si/alors, pour des systèmes a base de règles, des éditeurs de bases de connaissance permettent de représenter les règles pour les enrichir, modifier
moteur d'inférence : effecteur le raisonnement pour tirer le conséquences impliquées par la connaissance incluse (traiter le cas cause->effet)

interface : GUI, traitement de question réponse (ex : langage naturel), interface de consultation de système expert, interface pour l'acquisition de connaissances, mettre a jour et vérifier

Exigence raisonnement correct
raisonnement ouvert à l'inspection(etre capable d'expliquer pourquoi il a fait tel et tel choix)

Construction d'un système expert des modules pour la création d'un système expert peuvent être fournis ou réutilisés
CLIPS de la nasa , en C
JESS en java
ILOG rules de IBM
construction de la base de connaissance est polus importante et en générale plus difficile

10.2 Raisonnement en logique propositionnelle

la logique des propositions permet d'exprimer
des fait : jean aime marie
des negations : marie n'aime pas jean
ëes conjonction ou disjonctions
des phrases avec conséquence logique : si marie n'aime pas jean , marie friendzone jean
Une proposition es une expression phrase à propos du monde qui est vraie soit fausse
Elements de base : symbole de propositions, spéciales : vrai et faux, opérateur ANd et OR

$C_{i,j}$ breeze en ij, $P_{i,j}$ puit en i,j

P(2,3) -> $C(1,3) \cap C(3,3) \cap C(2,2) \cap C(2,4)$
Comme c'est bien de la merde, il n'est pas possible d'indiquer P(i,j) -> $C(i-1,j) \cap C(i+1,j) \cap C(i,j-1) \cap C(i,j+1)$, on est donc obliger de se faire chier à décrire toutes les cases, bienvenue dans le monde merveilleux des systèmes "expert"

sémantique Dasn une interprétation, un symbole sera remplacé par vrai ou faux
règles dévaluation : $\neg P$ est vrai ssi P est faux

base du Wumpus KB = l'ensemble de toutes les phrases en logique prépositionnelle décrivant le monde actuel

soit P_{ij} est vrai si un puits en ij, B_{ij} si breeze ne ij
pas de puits ne 11 $R_1 : \neg P_{11}$

résolution connaissance sous forme de clauses
chainage avant et arrière : connaissances sous forme de clauses de Horn(règles)

Résolution Exemple : on sais que p13 v p22, par perception et raisonnement on sais que ≠p22
grace à ce littéral on connais p22 = faux donc on résout p13, on enrichit donc notre connaissance du monde.
en plus formel , un littéral s=c'est un var ou sa neg , une clause c'est un disjonction de littéraux, la base c'est des conjonctions de clauses
règles de déduction $\frac{C_1 \wedge A, C_2 \wedge \neg A}{c_1 v C_2}$

Algo de résolution on cherche KB$\models \alpha$ par contradiction, montrons que $KB \models \neg\alpha$ est instatisfiable. on transforme en CNF pour résoudre, ajouter les nouvelles clauses produites, le processus continue jusqu'à ce qu'un des cas suivants se produise, plus de nouvelles clauses ou on trouve une clause vide

11 26 Mars 2013

la résolution ets correcte, si l'agent utilise la résolution pour interrgoer la base, alors les réponses sont correctes. Si on sais que KB U $\neg\alpha$ produit al clause vide , alors la claque α est une vrai conséquence des connaissances de la base . exemple si on hésite entre deux cases pour un puits , et que plus tard, on apprend qu'une de ces cases est vide, alors c'est l'autre qui a le puits.
LA complétude nous dit que si on a une conséquence, alors la résolution peut prouver que c'est une conséquence(la résolution arrête sur la clause vide)

clauses de Horn symbole de proposition $A_1 \cap ... \cap A_n -> B$
Règle de déduction :*Modus Ponen*(pour forme de Horn)

$$\frac{\alpha_1,...,\alpha_n, \alpha_1 \cap ... \cap \alpha_n -> B}{B}$$

$KB = B_{regles} \cap B_{faits}$

chaînage avant Chaque connaissance est une règle avec des prémisses et une conclusion
répéter jusqu'à saturation ou jusqu'à ce que l'expression à prouver soit trouvée

- activer les règles dont les prémisses sont satisfaites par des éléments de la KB
- ajouter la conclusion à la KB

Représentation de la KB sous forme d'arbre(graphe) ET-OU.
A chaque étape, on considère des règles, on regarde celles qui sont satisfaites,

on valide la conclusion on on l'ajouter aux faits , et on boucle jusqu'à ce que l'on ne puisse plus rien ajouter.

Chaînage arrière ON est orienté objectif, pour prouver Q, il faut prouver P , pour prouver P , il faut prouver L et M, pour prouver L , il faut prouver soit , P et A, soit A et B. soit la chaîne de preuve a chercher suivante :
Q
P
L,M
P,A,M ou A,B,M
A,B,M (on ne garde pas l'autre car elle boucle)
B,M
M
B,L
L(axiome)

ATTENTION : éviter les boucle, si un nouveau bu test déjà dans la pile de buts
éviter les taches répétées

le chaînage avant est piloté par les données(data-driven), un agent déduit des propriétés et des catégories à partir de séquences perceptives
il peux effectuer des taches non pertinentes par rapport à l'objectif
le chaînage arrière est en complexité

Construction de modèles preuve de satisfiabilité d'un ensemble de connaissances
$KB \models \alpha$ si et seulement si $KB \wedge \neg\alpha$ n'as pas de modèle. cette existence de modèle nous permet de résoudre α

Algo DPLL prendre ne entrée un ensemble de clauses
obj ; vérifier s'il existe une instanciation des symboles qui rend les clauses vérifiables
Algo : énum récursive à chaque étape, évaluer avec affectation partielle, comme c'est des disjonctions de conjonctions, une clause est vraie si l'un des littéral est vrai
si toute clause est vraie alors l'affectation partielle est un modèle, si une seule clause est non vraie, alors pas un modèle.

recherche locale pour trouver des modèles les algorithmes de recherche locales peuvent s'appliquer avec une bonne fonction d'évaluation, les états sont des interprétations possibles, la recherche locale cherche une interprétation qui est un modèle
Algorithme WalkSAT : à chaque étape, choisir une clause non satisfaite et un

symbole, changer la valeur du symbole. TRES EFFICACE si le modèle existe, si terminaisons en echec alors : pas suffisamment d'itération, soit pas de modèle du tout

Comment rendre notre agent prudent avec un algo de recherche : l'agent cherche a répondre à une question à l'aide de la base, pour savoir $KB \models W_{23}$ alors on cherche a repondre, $\exists modele : KB \cup \{W_{23}\}$

EXERCICE écrire la fonction pour le Wumpus
KB : base de connaissances
on suppose que l'on a deux fonctions :

- TELL(KB,α) : dit à la base que α ets dnas les faits et donc le rajouter (e.g : W23 : il y a un monstre en 2,3)
- ASK(KB,α) : demande à la base la question y'a t'il α

passons à l'algorithme du Wumpus : à chaque instant , il consulte ses connaissances, et doit répondre une action,

```
function Agent_Wumpus retourner action
Données :
KB : base de connaissances
breeze,smell,shine : trois sensors
x,y,dir // l'agent doit bouger
action : dernière action faite, initialement vide
dejaVues : ensemble des cases déja vues (sorte de chemin)
plan : plan d'actions à effectuer , initialement vide
```

pseudo-code

```
//maj
DejaVues <- ajouter(case(x,y))
accessible = accessible - case(x,y)
si action = aller_gauche
{
dir <- ouest
x<- x-1
}
si action = aller_droite
{
dir <- est
x <- x+1
}
//idem pour haut et bas
//----renseignement de la base
si odeur alors
{
TELL(KB,odeur(x,y))
```

```
}
sinon
{
TELL(KB, not odeur(x,y))
}
finsi

si courantdair alors
{
TELL(KB,courantdair(x,y))
}
sinon
{
TELL(KB,not courantdair(x,y)
}
finsi

si eclat alors
{
 retourner prendre_or
}
finsi

//---------------trouver une action
si plan d'action non vide
{
action <- 1ere action du plan
plan <- plan - 1ere action
}
sinon
{
tester les cases accessibles qui ne snot pas dejavues
accessible <- add voisins(x,y)
si on trouver une case "SAFE" C alors
si il existe une case(i,j) de accessible telle que
ASK(KB, notW(i,j) ^ notP(i,j)) == true;
{
calculer un chemin de notre position jusqu'à C
plan <- construire_chemin((x,y),(i,j),dejavu)
(parcours dans dejavu : largeur ou profondeur ou autre)
enrichir le plan d'action
effectuer la 1ere action
action <- sommet(plan) //car plan est une pile
deplier(plan)
}
sinon
```

```
{
pareil mais avec une case au hasard
action <- alea(listeAction)
}

}
retourner action
```

12 2 Avril 2013

12.1 Agents Logiques

la logique propositionnelle est déclarative : les éléments syntaxiques correspondent à des faits
Elle permet de décrire des info sous forme négative, conjonctive et disjonctive
Elle est compositionnelle.

12.2 logique du premier ordre

Elle permet de représenter des objets, contrairement à celle d'avant qui représente des faits, des faits sur les objets, la logique 1 ordre permet de représenter des objets dans des variables, comme dans le langage naturel. On peut donc définir des objets(personne, maisons, nombres, etc..), des relations(plus petit que, plus grand que, ... rond, gros, petit , grand, etc..), ainsi que des fonctions(le père, la mère, etc...)

Par exemple le successeur : D -> D (du domaine au domaine, par exemple d'un entier vers un autre entier)

le monde du wumpus en logique du premier ordre la KB est passée en premier ordre, TELL(KB,e) ajoute e à la KB, mais maintenant ASK(KB,S) retourne les substitutions σ telles que KB $\models$ Sσ, par exemple si on interroge sur W(x,y) pour connaitre ou sont les monstres, il nous répond les différents couple où il y a un monstre.

représentation de l'environnement perception au temps t. Percept([Smell/none,Breeze/none,Glitter/no
Actions : ici des termes, Turn(Right), Turn(left), forward, Shoot, grab, release
Cases : la case i,j est représentée par le terme [i,j]
relation d'adjacence : comme en C++ , mais avec le prédicat Adjacent([x,y],[a,b])
<=> $[a, b] \in \{[x + 1, y], [x - 1, y], [x, y + 1], [x, y - 1]\}$
Puit , prédicat unaire Pit(L) où L=[x,y]
Wumpus, constante, fonction Home(Wumpus) désigne la case de wumpus
Position de l'agent au temps t est al case s : At(Agent,s,t).

règles relatives au perceptions
$\forall b, g, t(Percept([smell, b, g], t) => Smelt(t))$
$\forall s, b, t, (Percept([s, b, GLitter], t) => AtGold(t))$
reflexe : $\forall t(AtGold(t) -> action(Grab, t))$
reflexe avec etat interne : as t on déja l'or ?
$\forall t(AtGold(t) \wedge \neq Holding(Gold, t)) => Action(Grab, t)$

Définition du prédicat Breezy : $\forall y Breezy(y) \equiv \exists x(Pit(x) \wedge Adjacent(x, y))$

Garder une trace de changements les faits n'ont pas une durée indéterminée, on rajoute donc un secodne param au prédicat pour indiquer l'instant

pour la loi : un crime pour un américain de vendre des armes a des nations enemy des US, Nono is pays enemey des US , armes sold by Col.West, who is american

$American(x) \wedge Weapon(y) \wedge Sell(x, y, z) \wedge Hostile(z) -> criminel(x)$ deduction : american(Col.West).
sold(Col.West,Nono).
enemy(Nono).
criminel(X) :- american(X),sold(X,Y),enemy(Y).
Skolemisation : remplacer une variable existentielle par une variable normale sans $\exists$ pour permettre l'unification, utile mais WHY ?

propriété du chainage avant
correct et complet, pour clause du 1 ordre
datalog(OMG ! ! ! !) clauses définies du premier ordre + sans fonctions.
Simple observation : à l'itération k , il n'est pas nécessaire de considéré un clause dont aucun des prémisses n'as été ajouté à l'iter k-1.
l'unification peut être couteuse
utilisée en KB déductive

chainage arrière procéder a partir du but, plus cool , on déroule les règles jusqu'à trouver des fait , ou etre bloqué, on peux l'effectuer en algo DFS, et donc espace utilisé linéaire, Incomplet à cause de boucle, utilisé en programmation logique.

Résolution pour résoudre $KB \models \alpha$ on ajoute $\neg\alpha$ dans la base

13 9 Avril 2013

13.1 Planification

definitions Partir d'une config initiale, arrivée à celle voulue et savoir comment on y est arrivé. atteindre un état final, satisfaisant des exigences, à partir de l'état courant et utilisant les actions possible(exemple , tourner le télescope

hubble) .
les recherches standards ne fonctionnent pas , car l'heuristique ne s'applique pas à tous les problèmes, donc nouvelle heuristique à chaque fois.

différence entre les deux approches

- représentation des états : RP : affectation des symbole pour chaque état, PL : expression logique
- représentation des buts : Rp : un symbole de but, PL : phrases caractérisant les objectifs
- Actions : RP : opérateur , d'un état à un autre, PL : adjonction/suppr de phrase représentant le monde
- Plans : RP : chemin,

la planification offre : langage uniforme (représentation logique), la possibilité d'ajouter des actions à un plan, n'importe ou, et pas forcément dans un ordre incrémental, possibilité de représenter que certaines parties du monde sont indépendantes.
planif par réduction de problème : il est plus facile de résoudre un problème complexe en le réduisant à un ensemble de sous-problème plus facile à résoudre.

calcul de situation Situation : état du monde à l'instant t.
Action : opérations permettant de changer de situations
les fluides sont des prédicats qui varient d'une situation à l'autre
on se dit : je dispose de trucs logique, je sais faire des enchaînements entre des clauses pour en produire d'autre . On rajoute Possible(a,s) qui est vrai si l'action a est possible dans la situation s.

-

13.2 STRIPS

les états sont représentés par : une conjonction d'atomes clos et sans symboles de fonction + CWA (closed world ass) , dans STRIPS on a le droit au $\neg$ seulement dans les effets des actions, pas le droit au négations pour décrire les faits, mais dans les effets le $\neg$ décrit le fait que l'on retire le fait de notre base de fait.

ADL :action description langage développe certains aspect que STRIP oublie

Algo pour la plannif problème de recherche sur des expression logique, donc chaînage avant ou arrière,
par contre les deux chaînages ne sont pas efficaces, dans STRIPS le coût d'une action est unitaire,, nous cherchons donc une heuristique .
La seule solution viable est est le problème relâché, nous cherchons donc à chaque fois vers une solution qui nous rapproche de la solution, le problème est que l'on supprime des contraintes et que du coup certains aspects bloquant sont ignorés(ex : colis indisponible s'il est dans un avion)

, on supprime les effets négatifs, on a donc tout le temps une augmentation de la base de fait. On pars aussi sur diviser pour régner .

Ordre partiel souvent il est superflu de classer deux suites d'actions qui n'ont aucun rapport entre elles par exemple : mettre des chaussures avec des chaussettes
n'importe quel ordre total que satisfait l'ordre partiel est solution. On pars d'un plan vide, on met ce que l'on connaît, début et fin.
on ne doit pas mettre de contrainte qui ajoute un cycle dans le graphe.
on peut aussi définir des liens de causalité $A- >^p B$ signifie que p est n effet de A et doit rester vrai entre a et B, on ne doit donc pas mettre entre ces deux étapes une action qui contient $\neg p$ dans ces effets.
On doit faire attention si on remonte que les préconditions générées par le chaînage sont toutes vraies quand on arrive au debut.

Exercice CT Disneyland représenter pb en STRIPS
$maison(A), maison(M), maison(E)$
poserEnfant :
PRECOND : maison(E)
EFFETS : chezPM(E), chezPM(A),chezPM(M) , ¬maison(E), ¬maison(A), ¬maison(M)

allerDisney(x) :
PRECOND : chezPM(x)
EFFET : ¬chezPM(x), aDisney(x).

allerAlice :
PRECOND(M)
EFFET : ¬aDisney(M), afaitAlice(M)

allerSpaceM
PRECOND : aDisney(A)
EFFET : ¬aDisney(A),afaitSpaceM(A)

allerPPan :
PRECOND : afaitSpaceM(A),afaitAlice(M),apasfaitPP(A),apasfaitPP(M)
EFFET : afaitPPan(A),afaitPPan(M),¬apasfaitPP(A),¬apasfaitPP(M)

allerSS :
PRECOND : afaitSpaceM(A),afaitAlice(M),apasfaitSS(A),apasfaitSS(M)
EFFET : afaitSS(A),afaitSS(M),¬apasfaitSS(A),¬apasfaitSS(A)

diner :
PC/ afaitPP(A),afaitPP(M),afaitSZS(A),afaitSS(M)
EF :dinerPris, ¬afaitPP(A), ¬afaitPP(M), ¬afaitSS(A), ¬afaitPP(M)

allerHT :
PC :dinerPris
EF : ¬dinerPris, afaitHT

TD7 -Barbie robot Init : NoSock(L),NoSock(R),NoShoe(L),NoShoe(R),NoHat,NoBag,InSide
End : OutSide

GoOut
PC : Shoe(L) $\wedge$ Shoe(R) $\wedge$ Hat $\wedge$ Bag $\wedge$ InSide
EF : $\neg$InSide,OutSide

WearSock(X)
PC : NoSock(X)
EF : $\neg$NoSocks(X),Sock(X)

WearShoe(X)
PC : NoShoe(X) $\wedge$ Sock(X)
EF : $\neg$NoShoe(X),Shoe(X)

WearHat
PC : NoHat
EF : $\neg$NoHat,Hat

TakeBag
PC : noBag
EF : $\neg$NoBag,Bag

Debut
PC : $\emptyset$
EF : Init

Fin
PC : End
EF : $\emptyset$

Exercice 4 : singe-banane

Init : lieu(singe,A),lieu(banane,B),lieu(boite,c),mobile(singe), poussable(boite), hauteur(singe,bas),hauteur(banane,haut), hauteur(boite,bas), prenable(banane),montable(boite)

aller(A,X,Y) :
PC : mobile(A), lieu(A,X), hauteur(A,bas)
EF : ¬lieu(A,X), lieu(A,Y)

Pousser(A,X,Y)
PC : lieu(singe,X), lieu(A,X), poussable(A), hauteur(singe, bas), hauteur(A,bas)
EF : ¬lieu(singe,X) ¬lieu(A,X), lieu(singe,Y), lieu(A,Y)

monter()
PC : lieu(singe,X), lieu(A,X), hauteur(singe, bas), hauteur(A,bas), montable(a)
EF : ¬hauteur(singe,bas), hauteur(singe, haut)