TDT4136 Introduction Artificial Intelligence Summary

Summary of material in TDT4136 Introduction to Artificial Intelligence

# Lecture 2 – Intelligent Agents

Agent Behavior

* Agent function – math description of behavior, maps percept sequence to and action
* Agent program – concrete implementation of agent function, runs withing a physical system

Doing the right thing rational agent

* Consequentialism – evaluates agents’ behavior by its consequences
* Performance measure – evaluates sequence of environment states
* Better to design performance measures according to achievement rather than how agent should behave

Rational agent maximizes its expected performance

* Performance measure
* The agent’s prior knowledge of the environment
* Action agent can perform
* Agents’ percept sequence to date

Rationality vs omniscient vs perfect

* Omniscient agent knows the actual outcome of its actions
* Perfect agents maximizes actual performance

Rationality requires information gathering and exploration

* Actions to modify future percepts

Rationality requires learning and autonomy

* Learning agents: modifies knowledge of the environment from experience
* Autonomous agents: learns what it can to compensate for partial or incorrect prior knowledge

Task Environment

* PEAS: Performance measure, environment, actuators, sensors
* Performance: safe, fast, legal, profit
* Environment: roads, traffic, customers
* Actuators: steering, display, speech
* Sensors: cameras, radar, microphones, touchscreen

Environment properties

Fully observable

* Relevant part of the state of the environment can be sensed
* No need to maintain any internal state to keep track of the wordld
* Chess, image analysis

Partially observable

* Parts of the environment cannot be sensed
* Agent make informed guesses about the world
* Poker, taxi driving, medical diagnosis

Single agent

* No other agents to influence my agent
* Medical diagnosis, image analysis

Multi-agent

* Agents’ performance depends on behavior from other agents
* Competitive and cooperative interactions
* Poker, chess, taxi driving

Deterministic

* Action has a singe guaranteed effect no uncertainty
* Chess, image analysis

Nondeterministic

* Given current state and action, some uncertainty about the next state
* Multiple outcomes, terms of probabilities
* Poker, taxi driving, medical diagnosis

Episodic

* Agent’s experience is divided into atomic episodes
* Each episode consist of agent perceiving and then performing a single action
* Choice of action in each episode depends only on the episode itself
* Image analysis

Sequential

* Current decision could affect all future decisions
* Poker, chess, taxi driving, medical diagnosis

Discrete

* Finite number of distinct states, percepts, and actions
* Chess, poker

Continuous

* Continuous time/state/actions
* Taxi driving, medical diagnosis, image analysis

Dynamic

* May change while an agent is deliberating
* Taxi driving, medical diagnosis

Static

* The environment does not change
* Poker

Semi dynamic

* The world does not change but the agent’s performance score may
* Chess with clock, image analysis

Known

* The agent’s knowledge about how the environment works/evolves
* The known environment may only be partially observable, even though it knows the rules

Unknown

* The agent will have to learn how it works
* An unknown environment can be fully observable

Agent = program + architecture

* The job of AI is to design an agent program that implements the agent function – the mapping from of percept sequence to action
* Agent architecture – computing device that runs the agent program with physical sensors and actuators

Table-driven agent

Text, letter

Description automatically generated

* The key challenge for AI is to find out how to write programs that to the extent possible, produce rational behavior from a small program, rather than a large table.

Simple reflex agent

Text, letter

Description automatically generated

* Acts according to rule matching the current state, handles fully observable environments.

Model-based reflex agent

* Maintains internal state
* Has a model of how the world works: transition and sensor models
* Handles partially observable environments

Text, letter

Description automatically generated

Goal-based agent

* Goal identifies desirable states
* What will happen if I do X
* More flexible, general than reflex agent
* Handles sequential environments, e.g., taxi driver, chess

Utility function

* Utility provides happiness level for sequence of states
* Same goal may be achieved in more than one way, with different utility values
* Utility function – internalization of the performance measure
* Expected utility – average utility weighted by state probabilities
* Handles nondeterministic and partially observable environments

Learning agent

* Becomes more competent than its initial knowledge allows
* Modifies its components to bring them closer agreement with the feedback
* Does some suboptimal actions to explore
* Handles well unknown and dynamic environments

True false?

* An agent that senses only partial information about the state cannot be rational: false
* There exist task environments in which no pure reflex agent can behave rationally: true
* There exists a task environment in which every agent is rational: true
* It is possible for a given agent to be rational in two distinct task environments: true
* Every agent is rational in an unobservable environment: false?
* A rational poker-playing agent never loses: false

# Lecture 3 - Solving Problems by Searching

Problem-solving process

* Goal formulation
* Problem formulation
* Search – simulate sequence of actions
* Execution

Problem formulation

* State-space, location on map
* Initial sate
* Goal state: isGoal(Bucharest) = true
* Actions
* Transition model
* Action cost function

Best-first search

Text

Description automatically generated

Node data structure

* Node.STATE
* Node.PARENT
* Node.ACTION
* Node.PATH-COST

Frontier operations

* IsEmpty(frontier): returns true only if there are no nodes in the frontier
* Pop(frontier): removes the top node and returns it
* Top(frontier): return but doesn’t remove the top node
* Add(node, frontier) inserts node in queue

Frontier data structures

* Priority queue – best first
* FIFO queue – breadth first
* LIFO queue (stack) – depth first

Dealing with redundant paths

* Cycle (loop) special case of redundant path
* Remember previously reached states (best first)
* Don’t worry about redundant path (assembly problem)
* Check for cycles but not for redundant paths
* Graph search vs tree-like search

Measuring problem-solving performance

* Completeness – is algorithm guaranteed to find a solution or failure
* Cost optimality – find solution with lowest cost
* Time complexity – how long does it take
* Space complexity – how much memory is needed

How to measure complexity

* Explicit: V + E: size of state-space graph
* Implicit:
* d. depth: number of actions in optimal path
* b. branching factor: number of successors of node to consider
* m. maximum depth of state space

Uniformed vs informed search

* Uniformed search: no clue how close state to goal
* Informed search: some estimate

Uniformed search strategies

* Breadth first
* Uniform cost
* Depth first
* Depth limited
* Iterative deepening

Breadth-first search

Text

Description automatically generated

* Complete: yes
* Cost optimal: yes, if actions have same cost
* Time and Space = O(b^d)

Uniform-cost search

* Handles varying positive step cost
* Uses priority queue sorted by path cost
* Expands nodes with least path cost first
* Complete: yes, if positive cost
* Cost optimal: yes
* Time and space O(b^1+(C\*/e))
* C\* optimal solution cost
* E – action cost lower bound

Depth first search

* Complete: No – fails in infinite depth spaces and spaces with loops
* Cost optimal: no
* Time O(b^m)
* Space O(bm) – linear

Iterative deepening search

Text

Description automatically generated

* Complete: yes
* Cost optimal: yes, if costs are equal
* Time: O(b^d)
* Space O(bd)
* Generally preferred if search state space is larger than memory and depth is not known

Uniformed strategies – summary of the properties

* b, branching factor
* d, depth of shallowest solution
* m, maximum depth
* l, depth limit

Table

Description automatically generated

Informed (Heuristic) search

* Use domain-specific hints about goal
* H(n) estimated cost of cheapest path

Greedy best-fist search

* Expand the first node with lowest h(n)
* Straight line to Bucharest
* Complete: Yes, in finite state space
* Optimal: No
* Time and space: O(b^m), with good heuristic can be O(bm)

A\* search

* f(n) = g(n) + h(n)
* g(n): path cost from initial node to node n
* h(n): estimated cost of the shortest path from n to goal state

Heuristic admissibility

* An admissible heuristic never overestimate the cost to reach goal
* Optimistic

Heuristic consistency

* Every consistent heuristic is admissible, but not vice versa

Properties of A\*

* Complete: Yes, for positive action costs, state space has a solution or is finite
* Cost optimal: for tree search with admissible heuristic, for graph search with consistent heuristic
* Time and space O(b^\*d) for good heuristic b\* < b

# Lecture 4 - Informed Search, Search in Complex Environments

Search problem

* State space
* Initial state
* Goal states
* Actions
* Transition model
* Action cost function

Informed search A\*

* F(n) estimated cost of the cheapest past through n to goal
* G(n) cost so far to reach node n
* H(n) estimated cost of the cheapest path from n to goal
* Complete yes for positive action costs in finite state space
* Cost optimal yes
* Finite branching factor
* Costs are positive
* Tree, h admissible and non-negative
* Graph, h is consistent and non-negative

Proof of optimality for A\*

* C\* star is optimal cost and A\* admissible heuristic returns a suboptimal path
* N is a node on the optimal path
* If A\* non optimal, n was not expanded
* F(n) > C\*
* For admissible h(n) we can derive f(n) <= C\*

Text, letter

Description automatically generated

f – contours

* A\* expands nodes in order of increasing f
* Surely expanded nodes
* Optimally efficient
* Pruning

Satisficing “good enough” solutions

* Inadmissible heuristic
* Overestimating optimal cost
* Risk missing optimal solution
* Potentially can be more accurate

Memory-bounded search

* Memory usage is an issue
* Reduce memory use
* Reference count
* Beam search, k best candidates
* Iterative deepening
* Recursive best-first search
* Memory bounded A\*

Bidirectional search – Uniformed

* Search forwards from initial state and backwards from goal, hoping they will meet

Text, letter

Description automatically generated

Learning to search

* Metalevel state space, internal state of a program
* Learning heuristics from experience

A picture containing text, watch, gauge

Description automatically generated

Search in complex environments

* Local search
* Search in non-deterministic environments
* Search in partially observable environments

Local search

* Path is irrelevant, goal state is the solution
* State space = set of complete configurations

Properties of local search

* Small memory uses
* Finds reasonable solutions
* Can solve optimization problems, finding best state according to objective function

Hill-climbing search

Graphical user interface, text, application

Description automatically generated

* Local maxima
* Ridges plateaus

How not to get stuck

* Sideways move
* Stochastic hill climbing
* First choice hill climbing
* Random restart hill climbing

Simulated annealing

* Idea: Escape local minima by allowing some bad moves, gradually decrease size and frequency

Text

Description automatically generated

Local beam search

* Idea: keep k states instead of 1, choose top k of all their successors
* Problem: often all k states end up on same local hill
* Idea: choose k successors randomly, bias towards good ones

Genetic algorithms

* Start with a population of k randomly generated states
* Randomly choose two parent states weighted by fitness
* Generate child by combining parent states randomly
* Mutate the child state with random changes
* Add child to population
* Repeat from 2 until child is fit or time elapsed

Recursive, depth-first and-or graph search

Text

Description automatically generated

# Lecture 5 - Adversarial Search and Games

Multi-agent, competitive environments

Board games

* Easy to represent state
* Small number of actions
* Precise rules

Physical games

* Complex descriptions
* Large number of actions
* Imprecise rules

Approaches in multi-agent environments

* Agents as aggregates
* Opponents are part of the environment
* Model agents explicitly

Two-player zero-sum games

* Deterministic, two-player, turn-taking
* S0 initial state
* ToMove(s)
* Actions(s)
* Result(s, a)
* IsTerminal(s)
* Utility(s, p)

Minimax search

* Conditional plan, similar to AND-OR search
* Goal, guaranteed win

Text

Description automatically generated

* Depth first, recursive
* Time complexity: O(b^m)
* Space complexity: O(bm) or O(m)

Alliances in multiplayer games

* Alliances – attack the strongest player
* Collaboration from selfish behavior
* Breaking alliance – trust issues
* Collaboration

Alpha-beta pruning

* Problem: O(b^d) – exponential number of states wrt to tree depth
* Solution: Prune (remove) parts of the tree that makes no difference in outcome for minimax decisions
* If m or m’ is better than for player, we will never get to n in play

Text

Description automatically generated

Alpha-beta pruning: Complexity

* Minimax with pruning: O(b^m) nodes
* Alpha-beta search (perfect ordering) O(b^m/2) nodes
* Alpha-beta search (random ordering= O(b^3m/4) nodes

Alpha-beta pruning: Move-ordering schemes

* Trying first the moves that were found to be best in the past
* From previous moves, same threats
* From previous stage of iterative deepening
* From transposition table, cache of heuristic values

Type A vs Type B strategy

* Type A: Wide but shallow – consider all moves to a certain depth, then heuristic function to estimate utility
* Type B: deep but narrow – ignores moves that look bad

Evaluation function characteristics

* Utility(loss, p) <= Eval(s, p) <= Utility(win, p)
* Strong correlation with chances of winning
* Fast

Evaluation function based on features and categories

* Define state features
* Define categories of equivalence classes of states with feature values
* Estimate proportion of states with wins losses and draws

Evaluation function based on combining feature values

* Material value for each piece
* Combine material values

Cutting off search

* In alpha beta replace is-terminal return utility with is-cutoff return eval
* Cutoff depth is fixed, iterative deepening

Forward pruning

* Remove moves that appear to be poor moves, but might possibly be good ones
* Beam search, n best
* ProbCut, alpha-beta with statistics
* Late move reduction, move ordering

Endgame lookup tables

* In combination with other approaches stock fish with depth 30

Weakness of heuristic alpha-beta tree search

* High branching factor can only reach shallow depth
* Hard to define good evaluation function

Monte Carlo tree search: Main idea

* Estimate value of state as the average utility over a number of simulations
* How to choose playout moves: playout policy

Monte Carlo tree search: Selection policy

* Exploration, states with few playouts
* Exploitation, states that have done well in past playouts

Text, letter

Description automatically generated

Monte Carlo pros and cons

Pros

* Better for high branching factor
* Less sensitive to evaluation errors
* Can be applied to new games without evaluation function
* Easily parallelized

Const

* Fails for games where a single move can often change the course of the game

Variation

* Monte Carlo tree search + evaluation function

AND-OR search for partially observable games

* Belief states
* Guaranteed checkmate
* Probabilistic checkmate
* Accidental checkmate
* Confusing random moves

Limitations of Game Search Algorithms

Limitations of alpha-beta

* Too slow for high branching factor
* Vulnerable to errors in heuristic function

Limitations of alpha-beta and Monte Carlo

* Don’t consider utility of node expansion: meta reasoning
* No abstract level reasoning with high-level goal
* Limited use of machine learning

# Lecture 6 - Constraint Satisfaction Problems

What is Constraint satisfaction problem (CSP)

* N-queens problem
* Place N queens on an NxN chess board with the constraint they don’t attack each other

N-Queen problem – backtracking

* Idea: Placing some queens, removes possible locations for the remaining queens
* Row by row
* If no column change column of last placed queen
* Got to 1

Applications

* Scheduling
* Assignment
* Configuration

Constraint satisfaction problem definition, components

* Variables
* Domains: Allowable values for each variable
* Constraint: Allowable combinations of values

CSP Map-coloring

* Color map with 3 colors so adjacent regions have different colors
* Variables {WA, NT, Q, NSW, V, SAT}
* Domain {Red, Green, Blue}
* Constraint, adjacent regions must have different colors

Constraint types

* Unary
* Binary
* Global
* Preference

Approaches to solving CSPs

* Inference, constraint propagating
* Incremental search, backtracking
* Search with inference, backtracking with forward checking
* Local search

Inference – constraint propagating

* Use constraints to reduce the number of legal values for variable
* Part of search
* Preprocessing step
* Sometimes solves the problem

Node consistency

* Node-consistent if every value in domain satisfies the variables unary constraint

Arc consistency

* Arc consistent if every value in domain satisfies the variables binary constraints

Text, letter

Description automatically generated

AC-3 Solution

* Solutions are assignments satisfying all constraints
* If one of the domains becomes empty
* Then no solution
* Else solution exists – all variables are arc consistent and no empty domains

Path-consistency

* Australia coloring with two colors, arc consistency does not help
* No valid assignments
* Higher order consistency, k-consistency

Global constraints

* All diff
* Resource constraints At most (10, p1, p2)
* Bounds propagation Between(10, 40, p1, p2)

CSP as search

* State-space: variable assignments so far
* Initial state: empty assignment
* Goal states: complete and consistent assignment
* Actions: variable assignment
* Transition model
* Action cost

Backtracking search

* Uninformed, depth-first search for CSP
* Variable assignments are communicative

Text

Description automatically generated

Improving backtracking using heuristics

* Which variable should be assigned next, var <- Select-unassigned-variable(csp, assignment)
* In what order should its values be tried, for each value in order-domain-values(csp.var.assignemnt) do
* Can we detect unpromising nodes early, inferences <- inference(csp, var, assignment)
* Can we take advantage of problem structure

Minimum-remaining-values (MRV) heuristic

* Which variable should be assigned next?
* Choose variable with fewest legal values
* Mos constrained variable, fail first
* Objective, detect immediately if variable has no legal values left

Degree heuristic

* Heuristic choose the variable with the most constraints of remaining variables
* Objective, reduce the future branching on future choices
* Helps in the beginning, tie breaker

Least-constrained-value heuristic

* Heuristic: given a variable, choose the value that rules out the fewest values in the remaining variable
* Objective: leave maximum flexibility

Improving backtracking using inference

* Detect unpromising nodes early
* Use AC-3 before or during search: time consuming
* Forward checking
* Arch consistency between unassigned neighbor nodes and current nodes
* Keep track of remaining legal values of unassigned variables
* Terminate when any variable has no legal values

Maintaining arc consistency (MAC)

* Inference with AC-3 starting with arcs to unassigned neighbor nodes
* Similar to forward checking but with propagation

Intelligent backtracking

* Chronological backtracking, change value for the most recent assignment
* Back jumping, change value for the most recent assignment in conflict set, redundant with forward checking and MAC
* Conflict-directed back jumping, updates conflict set to include conflicts with subsequent variables
* Constraint learning, find and avoid no-good partial assignments sets that cause the problem

Local search for CSP

* Start with complete assignment
* Randomly choose a conflicted variable
* Change its value so it brings us closer to a solution

Local search for CSP modifications

* Tabu search, helps with plateaus, remembers recent previous states
* Constraint weighting, focus on important often violated constraints

Local search for CSP performance

* Min conflicts is very effective for many CSPs

Problem decomposition

* Into independent subproblems
* Identifiable as connected components of the constraint graph

Tree-structured CSP

* Any two variables are connected by only one path
* Can be solved in time linear in the number of variables
* Topological sort
* All directed graphs consistent
* Use any remaining value

Reducing graph to trees – Cutset conditioning

* Remove subset of variables, called cycle cutsets, rest is a tree
* Solve tree CSP for each possible assignment of cycle cutset

Reducing graph to trees – tree decomposition

* Transform the original graph into a tree where each node consist of a set of variables

# Lecture7 - Propositional Logic

Knowledge-based agents

* Knowledge base, set of sentences that describe facts
* Inference engine, set of procedures

Operation on the Knowledge Base

* Add new knowledge
* Ask questions

Fundamental concepts of logic

* Syntax and semantics
* Possible worlds and model
* Entailment. A entails B, B follows from A, B is true whenever A is true

Computers and logic

* Natural language
* Knowledge representation language
* Programming language

Example – Compound sentences

* P: it is sunny this afternoon
* Q: it is colder than yesterday
* Lg: the traffic light is green
* Cg: the cars will go

Example – solutions

* It is not sunny this afternoon and it is colder than yesterday
* -P ^ Q
* If the traffic light is green, then the cars will go
* Lg -> Cg
* The cars will go only if traffic light is green
* Cg -> Lg

Truth-table enumeration algorithm

Text, letter

Description automatically generated

* Enumeration is sound and complete
* Exponential in the number of propositional symbols
* Model checking complexity
* Time complexity: O(2^n)
* Space complexity: O(n)

Theorem proving

* To build a proof of the desired sentence without dealing with model enumeration
* Proof is a chain of application of inference rules and logical equivalences

Entailment concepts

* α |= β and opposite, logical equivalence
* Validity, A sentence is valid if its true in all models
* Deduction theorem
* Proof by refutation or contradiction

Inference rules approach

* Check only entries for which KB is true
* Infer new logical sentences from the knowledge base and see if they match a query

Properties of an Inference Procedure and connection to Entailment

* Soundness: an inference procedure is sound whenever KB I- i a it is also true KB I= a
* Completeness: KB I= a also KB I-i a

Unsatisfiablility and proof by inference rule resolution

* Instead of KB I= a we show KB or not a is not satisfiable

Text

Description automatically generated

Conjunctive normal form

* A literal is either a propositional symbol or the negation of a symbol
* Every propositional sentence is equivalent to a conjunction of clauses
* Replacing a ⬄ b with (a => b) ^ (b => a)

Problem with resolution refutation

* Complete but can be exponential in time and space
* Reduce clauses to horn clauses it becomes linear
* Forward chaining and backward chaining

Horn clauses

* Definite clause: Disjunction of literals of which exactly on is positive: the rest is negative
* Horn clause: disjunction of literals of which at most one is positive
* Modus ponens is perfect for definite clause KBs

Forward and backward chaining

* Forward chaining, data driven
* Backward chaining, goal driven
* Both are complete for KBs in the definite clause form

Forward chaining

Text, letter

Description automatically generated

Backward chaining

* Idea: work backwards from query q
* Avoid loops
* Avoid repeated work where true or failed

# Lecture 8 - First Order Logic

Pros of propositional logic

* Declarative correspond to facts
* Allows partial information
* Compositional
* Context independent unlike natural language

Const of propositional logic

* Limited expressive power unlike natural language

Limitations of propositional logic

* Some knowledge is impossible to encode
* Relations and groups
* For example every student likes vacation, grows vary large
* Statement about similar objects and relations need to be enumerated
* If there is breeze in a square there must be pit in an adjacent square, this grows fast

How to say it in one sentence

* FOL, statement is represented by means of the following formula
* \-/square, adjacent(square, pit) => breeze

First order logic

* FOL assumes the world contains:
* Objects
* Relations
* Functions

FOL syntax

* Constants
* Predicates
* Functions
* Variables
* Connectives
* Quantifiers +-/ E

Universal quantifiers

* \-/ for all
* For all values of x, P(x) is true
* \-/ x At(x, NTNU) => smart(x)
* Everyone at NTNU is smart

Existential quantification

* E: there exists some
* E x at(x, NTNU) ^ smart(x)

Common mistake with quantifiers

* Typically => is the main connective with \-/
* Common mistake is to use and, this is wrong
* Typically E is the main connective E
* Common mistake is => this is wrong

Connections between \-/ and E

* All statements made with one quantifier can be converted into equivalent statements with the other quantifier by using negation
* ¬∃ ≡ ∀¬
* ¬∀ ≡ ∃¬
* ¬∀¬ ≡ ∃
* ¬∃¬ ≡ ∀

Multiple quantifiers

* More complex sentences can be formulated by multiple variables and nested quantifiers
* Variables must be introduced by quantifiers, and belong to the innermost quantifier that mention them

Order of quantifiers

* Reversing the order of the same type of quantifiers does not change the truth value of a sentence

Higher-order logic

* FOL allows quantifiers to range over objects but not properties, relations, or functions applied to the objects
* SOL allows quantifiers to range over predicates and functions as well

Wumpus in first-order logic

* Represent a square as an object s with coordinate functions x(s) and y(s)
* Represent square adjacency with a predicate

Reflex agent in the Wumpus world

* Rules that directly connect precepts to actions
* Requires a rule for each different combination of percepts, every possible state

Deducing hidden properties

* Diagnostic rule, infer cause from effect
* Causal rule, infer effect from cause

# Lecture 9 – Inference in First Order Logic

* Reduction of FOL to propositional logic
* Unification and First order inference
* Forward / Backward changing, on Definite Clauses
* Inference using Resolution, on CNF representation
* In FL we formulate the inference in the same way as propositional logic
* We want to find out whether a Knowledge Base entails a sentence alpha
* One inference approach, propositionalize the sentence in KB, and apply inference methods (resolution refutation) used in propositional logic
* Every KB in FOL can be propositionalized
* To propositionalize we need to get rid of quant ifiers and variables
* X is substituted with ground terms referring to objects
* Skolemization, skolem constant of skolem function
* Constant: if the existential variable is not within the scope of any universally quantified variable. Every instance of the existentially quantified variable. Every instance of the existentially quantified variable is replaced with the same unique constant, a brand new one that does not appear elsewhere in the knowledge base.
* Skolem function: if the existential quantifier is in the scope of a universally quantified variable, then replace it with a unique n-ary function over these universally quantified variables. Remove then the existential quantifier.
* Infinitely many ground terms: Father(Father(Father… )))
* Herbrand Theorem if a sentence is entailed by an FOL KB, it is entailed by a finite subset of propositionalized KB
* Propositionalizing is complete if the sentence is entailed
* Problem: works if a is entailed, loops if a is not entailed, hence it is semi-decidable
* Theorem: Turing Church, algorithms exist that say yes to every entailed sentence, but no algorithm exists that also say no to every non entailed sentence
* Propositionalizing seem to generate lots of irrelevant sentences
* With p k-ary predicates and n constants, there are p \* n^k
* Problem: Universal elimination gives us too many opportunities for substituting variables with ground terms
* Solution: avoid making blind substitution of ground terms
* Make substitution that help advance inferences
* Substitution matching similar sentences in KB
* UNIFICATION: takes two similar sentences and computes the unifier for them, that makes them look the same if it exists. UNIFY(P, Q) = Ø where SUBST(Ø, P) = SUBST(Ø, Q)
* Ground clauses are clauses with no variables in them. For ground clauses we can use syntactic identity to detect when we have a P and non P pair
* What about variables? For example, can the following two clauses be resolved?
* P(J) V Q (F) V R(x)
* ~P(y) V R(S) v R(y)
* Make substitution for values of variables that make the premise of the implication identical to the sentences in the knowledge base
* P1’ King(J) and P1 is King(x)
* P2’ Greedy(y) and P2 is Greedy(X)
* Generalized Modus Ponens is the lifted version of Modus Ponens
* Lifted inferences need finding substitutions that make 2 sentences look identical
* This is called Unification
* Unification process:
* Unify procedure: unify(P,Q) takes two atomic single predicate sentences P and Q returns a substitution that makes P and Q identical
* Aim is to match literals when they have variables
* Can replace variable by constant, variable, function expression without variable
* Not all formulas can be unified, only affects variables
* P(F(x), A) and P(y, F(w)) cannot be unified, no way of making A = F(w)
* Consider the two sentences: we want to keep the most general clause so that we can use it in future resolution steps
* Informally, the most general unifier (MGU) imposes the fewest constraints on the terms (contains the most variables)
* Formally, a substitution is more general thatn Beta if there is a substitution such that first second = beta
* Knows(John, x) and Knows(x, OJ) cannot be unified, needs standardizing apart
* Standardizing apart: eliminates overlap of variables

Inference through Forward Chaining in FOL

* Forward chaining is an important inference in FOL, without propositionalisation
* FC uses definite clauses, in FOL:
* Atomic or implication
* Existential quantifiers are not allowed
* Universal quantifiers are implicit
* Example:
* The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American
* We want to Prove that “Colonel West is a criminal”
* We need to:
* Translate these natural language sentences into OFL
* Convert sentences in the KB to definite clauses, Skolemization
* S1 It is a crime for an American to sell weapons to hostile nations
* American(x) and Weapon(y) and Sells(x, y, z) and Hostile(z) => Criminal(x)
* S2 Nono has some missiles
* Owns(Nono, M1) and Missile(M1)
* S3 all of the missiles were sold to it by colonel west
* Missile(x) and Owns(nono, x) => Sells(West, x, Nono)
* S4 Missiles are weapons
* Missile(x) => weapon(x)
* S5 an enemy of America counts as hostile¨
* Enemy(x, America) => Hostile(x)
* S6 American(West)
* Enemy(Nono, America)
* Checks every rule against every fact
* Rechecks every rule in every iteration
* Can generate many facts irrelevant to the goal

Backward chaining example

* Can solve Colonel West example using resolution refutation
* Two standardized clauses can be resolved if they contain complementary literals
* FOL literals are complementary if one unifies with the negation of the other
* Goal is to determine if KB |= alpha
* Convert KB to CNF
* Resolution rule to find out if satisfiable
* Question: Is colonel West criminal?
* Efficient resolution strategies:
* Unit preference: prefer inferences where one of the sentences is a single literal – towards shorter clauses, finally empty one
* Set up support: start with the negated query and add resolvents into the set
* Input resolution

# Lecture 10 – Knowledge Representation – and Ethical Problems in AI

Knowledge Representation

Ontological engineering

* Physical objects
* Actions
* Time
* Beliefs

General purpose vs special purpose ontology

* Should be applicable in more or less any special domain
* Add domain specific axioms
* Address all representations issues
* Different areas of knowledge needs to be unified
* Reasoning and problem solving could involve several areas simultaneously

Creating general purpose ontology

* By a team of trained ontologists or logicians
* By importing categories, attributes and values from existing databases
* By parsing and extracting information from text
* By crowsourcing, openMind
* Its hard to agree on ontology when many parties are involved

Categories and objects

* Much of the reasoning takes place at the level categories
* Categories help to make predictions about objects

Representation of categories and inheritance

* Predicate e.g. Apple
* Reify the category as an object
* Inheritance

Taxonomy hierarchy

First order logic and categories

* An object is a member of a category
* A category is a subclass of another category
* All members of a category have some properties
* All members of a category can be recognized by some properties

Natural kinds

* Some categories can be defined exactly, natural kind categories have no exact definitions

Relations between categories

* Disjoint: no member in common
* Exhaustive decomposition: all members are covered by categories
* Partition: an exhaustive decomposition of disjoint sets

Composite objects are often characterized by structural relations among parts. Biped is an object with exactly to legs attached to a body

Mental objects

* Knowledge about beliefs and deduction, useful for controlling inference
* Referential transparency, terms used don’t matter
* We need referential opacity

Semantic networks

* Semantic network is a type of logic
* Focus on connections between pieces of knowledge
* Efficient inference algorithm through inheritance
* Ability to represent default values for categories and allow overrides

Frames

* Contains slot with values
* Support inheritance and overrides
* Value can be another frame or procedure

Description logics

* Makes it easier to describe definitions and properties of categories
* Add formal logic based semantics to frames and semantic networks
* Tractability of inference

Description logics – inference

* Mechanisms:
* Subsumption, check if one category subset of another
* Classification, check if an object belongs to a category
* Consistency, check if membership criteria are satisfiable
* Problem solving steps:
* Describe a problem
* Check if it subsumed by solution categories

Reasoning with default information

* Commonsense reasoning is nonmonotonic
* Inheritance in semantic networks with default values
* Closed-world assumption

Circumscription

Default logic

* Use default rules for nonmonotonic conclusions

Truth maintenance systems

* KB contains P
* Tell (KB; -P)
* To avoid contradictions
* TMS revises beliefs to avoid contradictions

The Ethics of AI

Positive side of AI

* AI in crop management and food production
* AI in drug helps to cure diseases
* Drive assistance
* Optimization of business
* Automation for tedious and dangerous tasks
* Assistance for people with disabilities
* Smart grid and buildings
* Machine translation

Principles of robotics

* Ensure safety, fairness, privacy, collaboration, accountability

Lethal autonomous weapons

* Discrimination between combatants and non-combatants
* Judgment of attack necessity
* Assessment for potential collateral damage
* Reliability
* Cybersecurity
* Scalable weapons of mass destruction
* Attacker advantage

Lethal autonomous weapons, regulations, treaties

* 2014 ban of lethal autonomous weapons, supported by 30 nations
* 2015 campaign to stop killer robots

Surveillance

* 1976 warned speech recognition
* 2018 350 million surveillance cameras
* 2019 China social credit

Privacy

* Data collectors have a moral and legal responsibility to be food stewards of the data they hold
* Balance between privacy rights and society gains from sharing data
* Regulations

Privacy protection methods

* De-identification
* Generalization field
* K-anominity
* Aggregate querying
* Differential privacy
* Federated learning

Security

* High complexity, autonomy, dependance on data

Fairness and bias

* Designers of machine learning systems have a moral responsibility to ensure that their systems are fair
* Avoid discrimination by algorithms, bias in data

Fairness

* Individual fairness – similar individuals are treated similarly
* Group fairness – two classes are treated similarly
* Fairness through unawareness, remove gender and race fields
* Equal outcome – each demographic class get the same result
* Equal opportunity – individuals are treated according to their true ability, regardless of the class
* Equal impact – individuals with similar ability should have the same expected utility regardless of the class

Fairness and bias issues

* Biased data generated by biased societal process
* Ai is used to justify bias
* Finding the right fair objective
* Decide which classes to protect
* Sample size disparity
* Bias in software development process

Defences against bias

* Understand the limits of data
* Diversity among engineers
* De-bias your data
* ML algorithms that more resistant to bias
* Monitor metrics for different groups and fix bias issues

Trust and transparency

* Verification and validation methodology
* Certification of AI systems
* Transparency – explainable Ai

The future of work

* Technological unemployment
* Business process automation
* Income inequality

AI safety

* Failure models, what can go wrong
* Falut tree analysis
* Software engineering practices
* Utility function analysis – side effects, value alignment

Ultra-intelligence

* Technological singularity
* Thinkism
* Transhumanism
* Robopocalypse

# Lecture 11 – Automated Planning

Classical planning

* Find sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment
* So far:
* Problem solving by search
* Propositional logic
* Limitations:
* Require domain-specific heuristic or code
* Need explicitly represent exponentially large state

Factored representation for planning, Planning Domain Definition Language

PDDL – state

* Closed-world assumption – any fluents not mentioned are False
* Unique names assumption – different names refer to different entities
* No variables or negations
* Initial and goal state

PDDL – actions

* Action, precond, effect
* Ground action, action, precond, effect
* Any variable in the effect must also appear in the precondition

PDDL – Result of execution action

* A = Fly(p1, SFO, JFK)
* Del(a) = At(P1, SFO)
* Add(a) = At(P1, JFK)

Algorithms for classical planning

* Forward (progression) search
* Backward (regression) search
* Logical inference

Forward (progression) search algorithm

* Ground the actions by substituting any variable with constants
* Choose the ground action that is applicable
* Determine the new content of the knowledge base
* Repeat until goal state is reached
* Branching

Efficiency of forward search

* Forward search can have a very large branching factor

Backward (regression) search algorithm

* Pick a relevant action that satisfies (some of) goal propositions
* Make a new goal by applying an action backwards: Removing the goal conditions satisfied by the action, adding the preconditions of this action, keeping any unsolved goal propositions. Keeping any unsolved goal propositions
* Repeat until the goal set is satisfied by the start state

Relevant action in backward search

* A relevant action is one with an effect that unifies with one of the goal literal but with no effect that negates any part of the goal

Efficiency of backward search

* Lower branching factor than forward search
* Harder to come up with good heuristics because of variables in states

Planning as Boolean satisfiability (SAT)

* For each action substitute variables with constants
* Add action exclusion axioms to avoid two actions at the same time
* Add precondition axioms
* Define initial state, F0 for mentions and not F0 for not mentioned fluents
* Propositionalize the goal
* Add successor-state axioms

Other classical planning approaches

* Graph plan – encodes precondition and effect constraints in a planning graph
* Situation calculus – uses first order logic and first order solvers
* Constraint satisfaction problem – similar to SAT but more compact

Heuristics for planning

* Forward and backward search are inefficient without good heuristic
* How many actions are needed?
* We need admissible heuristic – don’t overestimate

Ignore-preconditions heuristic

* Every actions become applicable in every state
* Set-cover problem NP hard
* Greedy solutions not admissible
* Ignore selected preconditions

Ignore-delete list heuristic

* Relaxed version:
* Remove negative literals from preconditions and goals
* Remove delete lists from all actions
* Properties:
* No undo by other actions – monotonic progress
* NP Hard but can be approximated with hill-climbing

Domain-independent pruning – preferred action

* Focus on promising branches – how?
* Define and solve a relazed version to get relaxed plan
* Choose preferred actions – steps of the relaxed plan or achieve a precondition of the relazed plan
* Serializable sub goals:
* Order of subgoals that can be achieved without undoing previously achieved subgoals
* No backtracking!

State abstraction

* Many to one mapping of states to abstract representation
* Identifying and achieving similar subgoals as one, many packages with the same src and destination
* Creating heuristic form decomposed subgoals
* Pattern databases for subgoals

Planning and acting in classical environment

* Classical environment:
* Deterministic
* Agents knows effects of each action
* Fully observable
* Static
* The agent does not need perception:
* Can calculate which state results from any sequence of actions
* Always know which state it is in

Planning and acting in non-classical environment

* Non classical:
* Partially observable and/or nondeterministic environment
* Incorrect information (differences between world and model)
* The agents future actions will depend on future percepts
* The future percepts cannot be determined in advance
* Use percepts:
* Perceive the changes in the world
* Act accordingly
* Adapt plan when necessary

Types of non-classical planning

* Sensorless planning for environments with no observations
* Contingency planning for partially observable and nondeterministic environments
* Online planning and replanning for unknown environments

Sensorless planning

* No percepts – plans without seeing colors of paint, cans chair, table
* Open-world assumption, a belief state corresponds to the set of possible worlds that satisfy the formula representing it
* All belief states implicitly including unchanging facts
* Initial belief state includes facts that are part of the agents domain knowledge

Sensorless planning, use subset of belief state as heuristic

* Admissible, solving subset of a belief state is easier
* Combine heuristic values of individual states in belief state

Contingent planning

* For partial observable and/or non-deterministic environments
* Given a chair and a table, the goal is to have them of the same color. In the initial state we have two cans of paint, but the colors of the paint and the chair are unknown

Online planning

* Execution monitoring to determine the need for a new plan – replanning
* Incorrect model of the world: missing precondition, effect, exogenous events
* Action monitoring
* Plan monitoring
* Goal monitoring

# Lecture 12 – Multiagent Systems and Game Theory

Outline

* Micro vs macro aspects
* Strategical thinking
* Strategical normal games
* Game theory solutions, dominant strategy equilibrium
* Pareto optimal and social welfare

Agent design (micro) cs society design (macro)

* Micro questions:
* How do we build agents that are capable of independent, autonomous actions in order to successfully carry out the task that we delegate to them
* Macro questions:
* Types and characteristics of interactions between agents, interaction protocols
* Behavior in societies of self-interested agents

Game theory

* Studies the decisions of agents in a multiagent environment where:
* Actions of each agent have an effect in the environment
* So, outcome of an action of one agent may depend on other agents’ actions
* Hence strategical decision making

Strategical reasoning

* An agents decision depends on what it reasons/thinks the other agents will do
* The agent is uncertain about the decisions of the other participating agents
* Tries to predict others next decisions
* Computes how these affect him/her/them
* Makes own decisions accordingly

Let us play

* Choose number x between 0 and 100
* Calculate the average of all the numbers, the person number closest to 2/3 the average wins
* Everybody is rational

Sequential (extensive form) games

* Turn taking
* At each point the agents may decide/change their strategy
* Representation: tree
* Example: tik tak toe, chess, go

Strategic normal form games

* One shot
* The agents chooses its strategy only once at the beginning of the games, agent action simultaneously
* Main issue, predict other players action
* Represented: payoff matrix
* Prisoner dilemma, battle of the sexes, stag hunt

Assumptions of classical game theory

* Agents are rational. Well defined objectives, choose action with preferred outcome
* Agents have common knowledge. Rules of the game, know that other agents are rational

Defining strategical environments

* Agents
* Strategies/actions
* Outcomes
* Payoffs

Strategic normal games

* Agents N > 1
* Actions, chooses an action from set of actions
* Outcomes
* Utility function

State transformer function

* Each agent has just two possible actions that it can perform

Utilities and preferences

* Utility functions lead to preference ordering over outcomes
* Preferences determine selection of actions

Payoff matrix

* Agents utilities are shown in payoff matrix

Solution concepts

* A solution to a game is a prediction of the outcome of the game using the assumption that all agents are rational and strategic
* Strictly (strongly) dominant equilibriums
* Weakly dominant equilibriums
* Iterated elimination of dominated actions
* Nash equilibrium

Strictly dominant strategy equilibrium

* An action is strictly dominant if the agent prefers it over all of its other actions regardless of what actions the other agents prefer to choose
* A dominant action must be unique, and when it exists a rational agent will choose it
* Strictly dominant strategy equilibrium is the joint strategy where all the agents choose the strictly dominant action, if there exists

Weakly dominant strategy equilibrium

* Weakly dominant if it weakly dominates every action in set
* Neither agent has a strongly dominant strategy for both agents I and j
* Therefore b b is weakly dominant strategy equilibrium
* Rational agents will usually play this strategy

Iterated elimination of dominant strategies

* May be used where no dominant strategy equilibrium
* May be dominant action strategies
* A rational agent will never choose a suboptimal action
* IESDS solution technique that iteratively eliminates strictly dominated actions from all agents until no more actions are strictly dominated
* If no strongly dominated actions, use weakly dominated ones to eliminate
* Problems:
* Different outcomes may arise as outcomes that survive
* May be left more than one solution
* Need stronger solution concept: Nash equilibrium

Nash Equilibrium

* Pareto optimality:
* There is no other outcome where some agents can increase their payoffs without decreasing the payoffs of other agents
* Social welfare:
* Sum of utilities of all agents for this strategy profile
* If a solution maximizes social weldfare, social optimum, then the available utilities are not wasted
* if social optimum, then pareto efficient
* Nash quilibrium is a strategy profile, one for each player, such that each strategy is a best response to all other agents strategies
* How to find:
* Utilize best response correspondence in payoff matrix
* Mark best respone of each agent given the action choice of the other agent.
* Multiple Nash Equilibria:
* Not every interaction scenario has a single nash equilibrium
* Consider two agents each choose either left or right. If choices don’t match payoff is zero

Stag hunt – solution selection

* Conflicting opinions
* Harsanyi and selten 1988 propose payoff dominant equilibrium as rational choice
* Harsanyi 1995 changes conclusion take risk dominance

Why inefficient outcomes becomes solution?

* Need for new theories
* Need for modification in the modelled environment

# Lecture 13 – Summary